PolitMarket

SVP SP FDP CVP Grüne JA NEIN

Diplomarbeit	**September 2005 – Oktober 2005**
Studiengang:	**Kommunikation und Informatik**
Studenten:	**Micha Rieser** **Giancarlo Scrugli**
Betreuende Dozenten:	**Reto Ferri** **Prof. Dr. Hans-Peter Hutter**

Zürcher Hochschule Winterthur

I. Abstract (deutsch)

Ziel: Wir haben eine Webapplikation Prognosebörse, optimiert für Schweizer Abstimmungen und Wahlen, entworfen und implementiert. Die Idee ist, dass auf kommende Ergebnisse von Wahlen und Abstimmungen Aktien herausgegeben werden. Aufgrund des Prinzips eines Wertschriftenmarktes, werden diese durch den Handel direkt und fortlaufend bewertet. Daraus resultiert eine Prognose für den Abstimmungs- bzw. Wahlausgang.

Technologieentscheide: Die Prognosebörse besteht aus einer 3-Tier-Architektur mit .NET-Technologie. Als Datenbanksystem setzten wir MS SQL ein. Für die Business-Logik C# und für die Web-GUIs ASP.NET.

Projektvorgehen: Das Projekt bestand aus acht Iterationen (eine Iteration pro Woche). In der ersten Iteration legten wir das gesamte Projektvorgehen fest und definierten die Anforderungen an das Gesamtsystem. In den drei darauf folgenden Iterationen haben wir die Business-Logik entworfen, implementiert und ausgetestet. In den nächsten drei Iterationen haben wir dann die Web-GUIs entworfen und implementiert. In der siebten Iteration gingen wir in einen Testbetrieb mit echten Händlern, der bis zum 27. November 2005 unter http://www.politmarket.net fortgesetzt wird. In der letzten Iteration vervollständigten wir die Dokumentation und überwachten den Testbetrieb.

Resultat: Wir sind mit dem Projektergebniss zufrieden. Wir haben einen funktionierenden Betrieb aufbauen können. Wir haben folgende Punkte realisiert:
- Transaktionsgesteuerter Kern
- Datenbank-Entwurf
- ASP.NET-GUIs mit eigenen Webcontrols
- Prozess für Generierung von Statistikgraphiken
- Testbetrieb auf eigenem Server mit eigener Domäne (www.politmarket.com/.net/.ch)
- Webauftritt mit eigenem Corporate Design.
- Dokumentation (alle SWE-Artefakte und Hintergrundsinformationen)

Folgendes haben wir teilweise implementiert und sind als geschützte Variationen vorhanden:
- Mehrsprachigkeit (realisiert bei ASP.NET-WebControls)
- Eigene Protocoll-Klasse (für Logging)
- Portfoliohandel-Robot

Weitere Ausbauschritte: Um das Projekt weiterzuverfolgen haben wir bereits Ausbauschritte nach der Diplomarbeitsabgabe und -bewertung überlegt:
- Administrationstool für einfaches Monitoring (Logs, Auftragsbuchsicht pro Auftrag, Depotsicht pro Händler, Editieren von Benutzerdaten)
- Administrationstool für einfacheres Generieren neuer Abstimmungen und Wahlen.
- Content Management für News.
- Unterstützung von redundanten Datenbank-Servern und Backup-System.
- Ausbau von Statistiken (pro User und im Vergleich zu anderen Abstimmungen)
- Ausbau von Informationen (Geographische Karten von Abstimmungen und Verteilung der Benutzer in der Schweiz.)

Folgende Verbesserungen an unserem System wurden durch den Testbetrieb evident:
- Editierbarkeit der Aufträge
- Kaufpreisberechnungen in der Depotübersicht
- Nachführung von Portfoliokäufen und –verkäufen

I. Abstract (english)

Aim: To design, develop and implement a web based forecast stockexchange application. The idea is for shares to be issued according to the results from elections. Based on the securities trading principal, these shares will be evaluated through direct and continuous trading, which consequentially results in a forecast for outcome of the election.

Technological decisions: The forecast stockexchange will consist of a 3-tier architecture with .NET technology. The database shall be supported using MS SQL, the business logic with the programming language C# and the web user interfaces with ASP.NET.

Procedure: The project broken down into 8 iterations (one iteration per week). In the first iteration we determined the complete project procedure and defined the requirements for the whole system. In the following 3 iterations we developed, implemented and tested the business logic. In the next following 3 iterations we developed and implemented the web user interfaces. In the seventh iteration we moved into a live test environment, with real traders, which will remain online until 27th November 2005 (http://www.politmarket.net). In the final iteration we completed our documentation and monitored the test environment.

Result: We are extremely satisfied with the outcome of the project. We were able to develop a functional and working application. The following points were implemented:
- Transactional controlled core
- Database concept
- ASP.NET user interfaces with their own web controls
- Process for generating statical graphs
- Test environment with its own server and domain (www.politmarket.com/ .net/ .ch)
- Homepage with own corporate design
- Documentation (all software development artefacts and background information)

The following we were partially able to implement and are available as a protected variation:
- Multilingualism (implemented using ASP.NET web controls)
- Own protocol class (for logging)
- Trading portfolio robot

Upgrading: In order to pursue the project further, we have already considered steps for upgrading the application after the completion of the thesis:
- Administration Tool for simple monitoring (Logs, transaction overview per transaction, deposit overview per trader, editing of user data)
- Administration Tool for generating simple elections
- Content Management for news
- Support for redundant database servers and backup systems
- Upgrading of statistics (per user and in comparison to other elections)
- Upgrading of information (geographical map of elections and distribution of users in Switzerland)

Through the test environment, the following improvements have proven to be required:
- Editing of transactions
- Purchase price calculation in the deposit overview
- Tracing of portfolio sales and purchases

II. Erklärung betreffend das selbständige Verfassen der Diplomarbeit im Departement T

Mit der Abgabe dieser Diplomarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig ausgeführt hat, und dass sie keine Beiträge Anderer enthält, die nicht mit einer Quellenangabe versehen sind (bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Diplomarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 41 und 42 (Unredlichkeit und Verfahren bei Unredlichkeit) des Reglements für Prüfungen am TWI sowie die Bestimmungen des Disziplinarverfahrens der Hochschulordnung in Kraft.

Ort, Datum: Unterschriften:

Zürich, 26.10.2005

Micha Rieser

Giancarlo Scrugli

Inhaltsverzeichnis

I	Abstract	2
II	Erklärung betreffend das selbständige Verfassen der Diplomarbeit im Departement T	4
1	Aufbau des Dokuments	6
2	Aufgabenstellung	7
3	Theoretische Betrachtungen	8
3.1	Fragestellung	8
3.2	Halloween-Masken	9
3.3	Iowa Presidential Stock Market	10
3.4	Funktionsweise Prognosebörse	11
3.5	Motivation für unsere Prognosebörse	11
3.6	Gegenüberstellung von Umfrage und Prognosebörse	12
4	Projektmanagement	15
4.1	Prozess-Modell	15
4.2	Verlauf	16
5	Analyse	19
5.1	FURPS+	19
5.2	Die 3-Tier-Architektur	20
5.3	Restriktives vs. offenes System	21
5.4	Das Domänen Modell	24
5.5	Anwendungsfälle	25
5.6	Beans Konzept	40
6	Design	41
6.1	Typische Transaktion in der Klasse TransactionController	41
6.2	Kollaborationsdiagramm	48
6.3	Klassendiagramm	50
6.4	Klassen des Kerns	51
6.5	View-Klassendiagramm	55
6.6	Klassen der View	56
6.7	Der Datenbankaufbau	64
7	Implementierung	64
7.1	Sicherheit	64
7.2	Web-Controls	65
7.3	Testbetrieb	66
8	Test	69
8.1	Blackbox-Test OrderBook	69
9	Quellenverzeichnis	75
10	Anhang	76
10.1	Detaillierter Iterationsplan	76
10.2	Glossary	77
10.3	Benutzerhandbuch	79
10.4	Theoretische Abhandlung der Börse aus der PA2	91
10.5	Sitzungsprotokolle	97

1. Aufbau des Dokuments

Die Dokumentation ist in zwei Teile aufgeteilt. Einerseits haben wir theoretische Betrachtungen und Hintergrund-Informationen zum Thema zusammengestellt. Andererseits den umfangreichen Teil der Softwareentwicklung, wo Sie alle Artefakte zu den verschiedenen Disziplinen finden.

Das Kapitel „Theoretische Betrachtungen“ soll Sie dem Thema Prognosebörse näher bringen und Ihnen zeigen, worin unsere Motivation zu dieser Diplomarbeit bestand.

Im Hauptteil der Dokumentation können Sie die Entwicklungsarbeit nachlesen. Der Code zur Diplomarbeit befindet sich in der beigelegten CD-ROM.

Im Anhang sind weiterführende Informationen zur Börse, so auch der detaillierte Iterationsplan und das Glossary. Die Sitzungsprotokolle befinden sich am Schluss des Anhangs.

2. Aufgabenstellung

Schlussdiplomprüfung Praktische Prüfung

Studiengang:	Kommunikation und Informatik	**Jahr:** 2005
Experte/Expertin:	U. Thöni D. Rutishauser	

Studierende:	Micha Rieser, Giancarlo Scrugli
Dozenten:	Reto Ferri Prof. Dr. Hans-Peter Hutter
Klasse:	KI3d

Ausgabe der Aufgabe: 2. September 2005

Abgabetermin: 28. Oktober 2005

Thema: Prognosebörse
Aufbauend auf einer früheren Arbeit [RiSc05] soll eine Prognosebörse entwickelt und implementiert werden. Bei dieser Prognosebörse werden eine Art Optionen auf Abstimmungs- beziehungsweise Wahlresultate gehandelt. Der sich einstellende Börsenkurs der einzelnen Wahlergebnisse widerspiegelt die aktuellen Erwartungen an den Wahlausgang und erlaubt damit eine Prognose.

Anforderungen
- Die Prognosebörse soll sowohl für Abstimmungen als auch für Wahlen einsetzbar sein.
- Der eigentliche Handel der Optionen soll möglichst realitätsnah implementiert werden.
- Neben der Berechung der Börsenkurse soll die Börse auch für jeden Mitspieler eine persönliche Statistik ausgeben (Gewinn/Verlust)

Aufgaben
- Erheben Sie die detaillierten Anforderungen an die Prognosebörse.
- Entwickeln Sie sodann ein Konzept für das gesamte Börsensystem und implementieren Sie es.
- Testen Sie das entwickelte Börsensystem in geeigneter Form.
- Dokumentieren Sie Ihre Arbeit und die Testresultate ausführlich.

Quellen
[RiSc05] Micha Rieser, Giancarlo Scrugli: Börsensimulation, Projektarbeit ZHW, Studiengang KI, Winterthur, Juli 2005.

3. Theoretische Betrachtungen

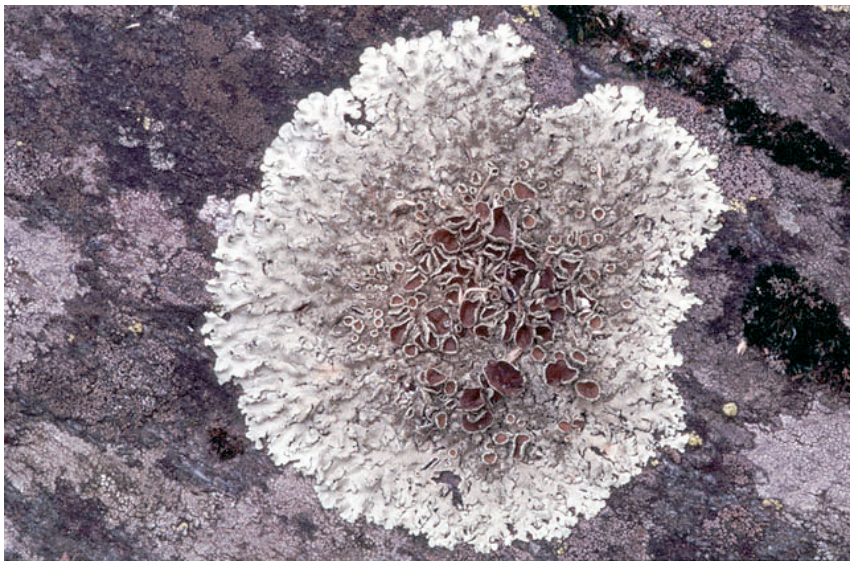
3.1. Fragestellung

Die Fragestellung unserer Diplomarbeit ist die folgende: Wie kann man eine politische Prognose machen, welche Prognosequalität über diejenige von Umfragen hinausgeht. Zuerst müssen wir uns deshalb fragen, wie überhaupt Prognosen erstellt werden können. Wenn wir die wissenschaftlichen Methoden ansehen, wie Prognosen gemacht werden, sehen wir verschiedene Beispiele, die nichts mit Befragungen von Zielgruppen zu tun haben. Vor allem wenn es keine Zielgruppen gibt oder die Zielgruppen schwer zu finden sind, dann setzt die Wissenschaft auf andere Indikatoren. Bei der Wirtschaft sind dies z.B. nun Arbeitslosenzahlen, Konsumentenstimmung, Inflation, etc. Bei der Politik sind dies z.B. Staatsquote, Sorgenbarometer, Wahlbeteiligungen, Alterserwartung, etc.

Es ist also so, dass nicht nur direkte Befragungen, gute Ergebnisse liefern, sondern auch indirekte Erscheinungen.

Dazu einen kleinen Exkurs in die Biologie:

Man kann die Luftqualität messen, indem man die Schadstoffe herausfiltert und chemisch analysiert. Um eine Luftkarte zu erstellen, müsste man also regelmässig und auf einer festgelegten Matrix jeweils dieselbe Messung durchführen. Es hat sich aber gezeigt, dass Flechten (Symbiose zwischen Pilzen und Algen) sehr empfindlich auf die Luftqualität reagieren. Flechten sind also ein indirekter Bioindikator auf die Luftqualität. Es werden nun regelmässige Zählungen von Flechten in einem bestimmten geographischen Raum durchgeführt und man erhält so einen guten Überblick über die Luftqualität in diesem Raum. [1]



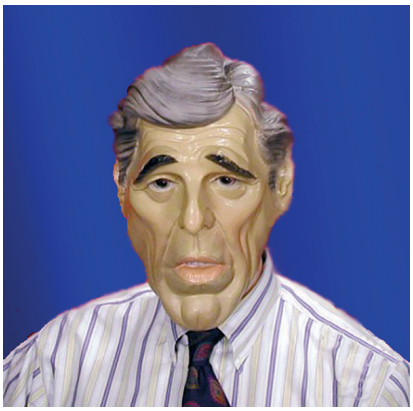
Im „Forschung und Technik"-Teil der NZZ wurde am 19. Oktober berichtet, dass vor allem die USA unter dem West-Nil-Virus leiden. Um festzustellen, wie verbreitet nun dieser Virus ist, bedienen sich die Staaten Florida und New York zwei unterschiedlicher Indikatoren. Florida überprüft über Satelittenbilder die Grünflächen, um darüber Auskunft zu erhalten, wie gross die Wasserflächen gerade sind. Der West-Nil-Virus wird über Stechmücken übertragen, die sich in den Wasserflächen vermehren können. Die Schlussfolgerung ist deshalb sehr einfach: Mehr Grünflächen → mehr Wasser → mehr Mücken → mehr Virenerkrankungen. Obwohl diese Kette über drei Umwege führt, liefern die Messungen sehr genaue Prognosen. New York dagegen zählt die Meldungen

von aufgefundenen Krähenleichen. Die Vögel sind sehr empfindlich gegenüber dem Virus und verenden deshalb sehr schnell. Wenn nun die Zahl der Krähenleichen zunimmt, ist das ein guter Indikator für eine höhere Virenzahl.

Einen ganz besonderen Indikator für Präsidentschaftswahlen finden sie im nächsten Kapitel.

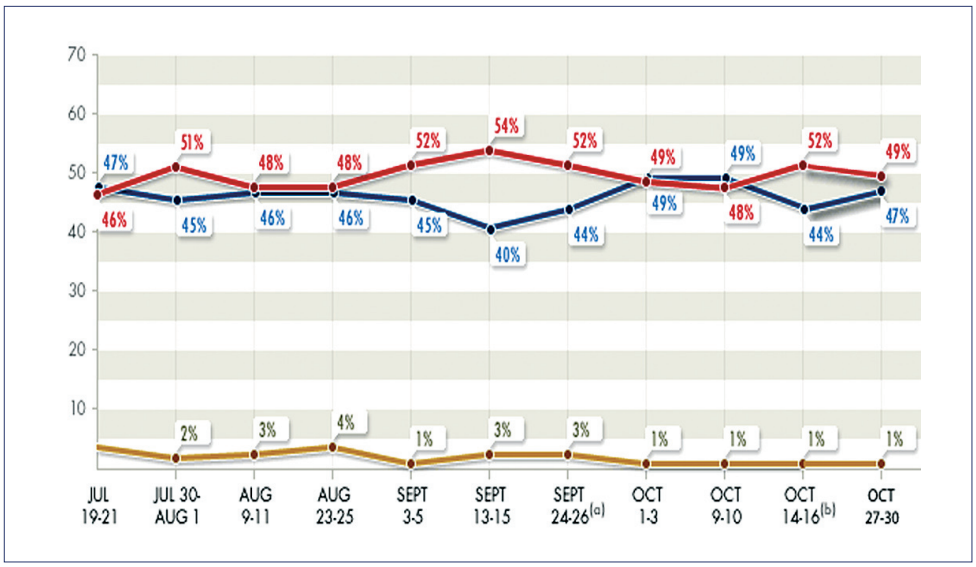
3.2. Halloween-Masken

Da wir Halloween haben, wenn wir die Diplomarbeit abgeben, möchten wir hier zuerst einmal auf eine sichere Wahlprognose in der USA eingehen: Halloween-Masken. Da die Präsidentschaftswahlen in den USA kurz nach Halloween sind, decken sich die US-Bürger gerne mit Masken der Kandidaten ein. Beliebter sind dann aber seit der Wahl von Reagan die Masken der zukünftigen Gewinner und so wurden die Verkaufszahlen der Masken zur ungewöhnlichsten aber sehr populären Wahlprognose der USA, die seit dem Sieg von Reagen stets Recht behalten hat. Warum die US-Bürger so verlässlich diese Masken kaufen, liegt im Dunkeln. Entweder hat der einzelne Amerikaner ein gutes Gespür, wie er (mit Maske) stets als Sieger dasteht, oder er gruselt sich an der Regierung des zukünftigen Präsidenten immer so sehr, ganz egal, wer das Amt nun innehat. An letzt jährigem Halloween (31. Oktober 2004) verkauften sich die Masken von George W. Bush zu 55%. Im Gegensatz kamen die Masken seines Herausforderer John Kerry nur auf 45%. Der Sieg war George W. Bush nicht mehr zu nehmen, obwohl die CNN-Umfragen dort noch ein Kopf-an-Kopf-Rennen prognostizierten.



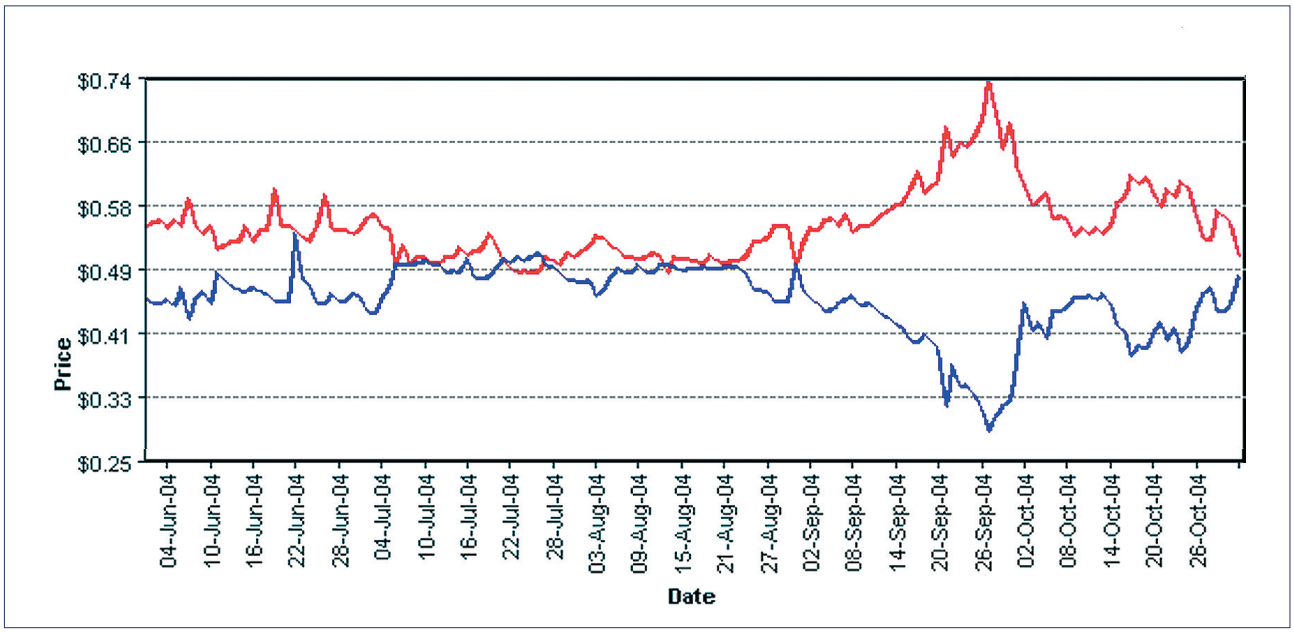
Die USA war wohl immer Ursprungsland für ungewöhnliche Wahlprognosen. Es verwundert deshalb nicht, dass auch die Idee, man könne mit Wahlprognosen handeln, wie mit Aktien, aus dem Land der unbegrenzten Möglichkeiten stammt.

3.3. Iowa Presidential Stock Market



CNN-Umfragen

An der University of Iowa wurde 1988 zum ersten Mal ein elektronischer Prognose-Aktienmarkt implementiert. Er trug die Bezeichnung IPSM (Iowa Presidential Stock Market) und diente für die Prognose der Stimmanteile der Präsidentschaftskandidaten. Aktien waren „Bush“, „Dukakis“ und „Sonstige“. Die Prognosegenauigkeit überstieg die aller veröffentlichten Umfrageergebnisse. Sie lag gerade mal 0.2% vom Endergebnis entfernt. (Wahrscheinlich Zufall, zukünftige Prognosemärkte zeigten, dass sie sich auch durchaus irren können. Ebenso ist die Prognose bei einem Aktienmarkt ein Prozess, es sind nicht die Endkurse interessant, sondern vor allem die Kursentwicklung.) Seitdem wird das System in anderen Ländern wie z. B. Deutschland, Dänemark, der Türkei, Kanada, Österreich und der Schweiz (Nationalratswahlen 1999 und



Iowa Electronic Market

2003) für Wahlprognosen eingesetzt. Ebenso wird versucht auch andere Prognosen wie z.B. Prognosen für Arbeitslosigkeit mithilfe von elektronischen Aktienbörsen zu erstellen.

Die Iowa Presidential Stock Market werden seit 1988 aber nun jede Legislatur durchgeführt. Einen Vergleich von den letzten Präsidentschaftswahlen zwischen der IPSM und der CNN-Umfrage finden sie hier.

3.4. Funktionsweise Prognosebörse

Es ist deshalb immer wieder interessant, neue Indikatoren zu finden, die uns eine Prognose ermöglicht. Die Börse organisiert den Sekundärmarkt von Aktien (Primärmarkt ist die Ausgabe der Aktien durch die Firmen selbst). Die Aktien werden herausgegeben um finanzielle Mittel in eine Firma zu bringen. Dagegen erhält der Shareholder Stimmgewalt und Eigentumsrecht an der Aktiengesellschaft. Die Firma muss aber die Aktien nicht zurückkaufen und so hat ein Aktienbesitzer keine Möglichkeit sein Geld wieder herauszulösen. Er kann aber einen anderen Shareholder finden, der für ihn in die „Lücke" springt und die Aktien hält. Da die Firma über die Jahre aber unterschiedliche viel Wert hat und die Aktien ja ein Eigentumsrecht darstellen, müssen sich der scheidende und der zukünftige Shareholder auf einen gerechten Preis einigen. Die Börse anonymisiert nun aber die Suche nach dem neuen Shareholder und löst das Problem der Preisfindung über ein System. (Lesen Sie dazu das Kapitel Auftragsbuch im Anhang nach.) Aus dem gehandelten Preis (Kurs) lässt sich also schlussfolgern, wie viel Wert die Summe der Anleger einer Aktiengesellschaft beimessen. Genau dieses System der Wertbemessung kann man nun brauchen, nicht nur um Firmen zu bewerten, sondern auch andere Dinge, wie z.B. kommende politische Ergebnisse.

Über den Marktmechanismus werden also die verteilten Informationen der Händler zusammengeführt. Der ökonomische Gewinn/Verlust dient als Anreizsystem um die eigenen Informationen in den Markt einfliessen zu lassen und über ihn zu reflektieren. Die gehandelten Aktien sind mit Optionen zu vergleichen. Die Höhe der Auszahlung nach Handelsschluss wird durch eine durch eine *Liquidationsformel* festgelegt. Normalerweise ist die Formel, dass sich der Wert proportional zum Abstimmungsergebnis in Prozent verhält. Es ist aber auch eine Formel nach dem Prinzip „the winner takes it all" denkbar. Aufgrund der allen Teilnehmern bekannten Liquidationsformel und der Aussicht auf Gewinn muss jeder Händler versuchen, geschickt zu agieren und so seine Strategie, seine Risikobereitschaft und seine Vorlieben festlegen.

Ein Händler gibt so Verkaufs und Kaufauträge auf und nach den festgelegten Börsenregeln kommen bei Überschneidungen Abschlüsse zu Stande, die dann den Kurs (die Prognose) bestimmen.

Da sich keine Blasen bilden dürfen (alle Aktien einer Abstimmung prognostizieren mehr als 100% zusammen), ist es möglich, am Markt jeweils ein Portfolio* zu 100 zu kaufen oder zu verkaufen. Dadurch pendelt sich der Markt automatisch immer auf die 100 ein.
* bei einer Vorlage wird pro Sorte je eine Aktie genommen und als Portfolio zusammengefasst.

3.5. Motivation für unsere Prognosebörse

Die Prognosemärkte in der Schweiz wurden für die Nationalratswahlen 1999 und 2003 durchgeführt. Sie gaben schlussendlich die Idee eine eigene Prognosebörse zu entwickeln um folgende Aspekte besser zu berücksichtigen:

1. Der Prognosemarkt soll nicht nur bei Nationalratswahlen eingesetzt werden, sondern auch bei Abstimmungen.
2. Der Prognosemarkt für Wahlen soll nicht bloss drei Monate vor Nationalratswahlen eingesetzt werden, sondern auch währen der Legislatur laufen, um fortwährend die Entwicklung in der Parteienlandschaft abzubilden.
3. Soll mehr Wert auf die Prognosequalität gelegt werden, indem Informationen für die Händler aufgearbeitet und auf der Plattform zusammengeführt werden. (Informationen wie Neuigkeiten, Umfrageergebnisse, vergangene Abstimmungsergebnisse, etc.)
4. Das Ganze soll unter einer Non-Profit-Organisation betreibbar sein, so dass der Betrieb nicht abhängig ist vom Interesse und dem finanziellen Einsatz von grösseren Medien.

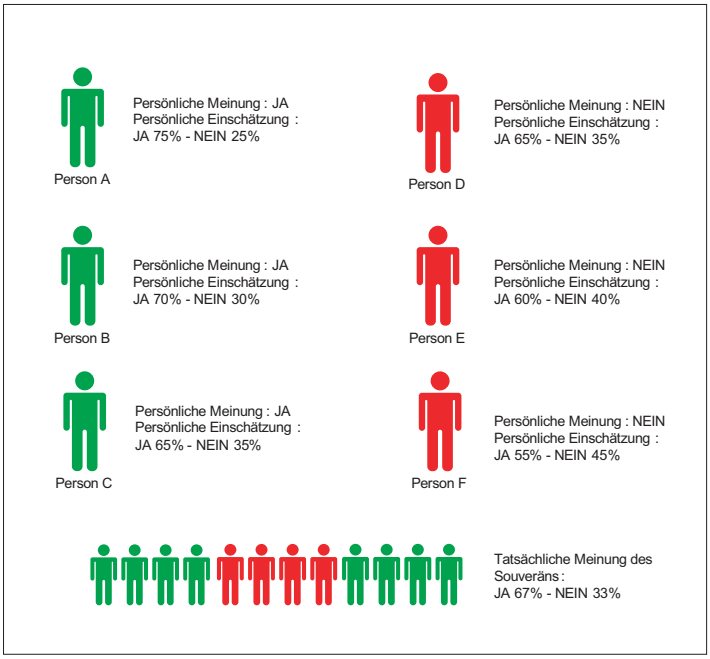
3.6. Gegenüberstellung von Umfrage und Prognosebörse

3.6.1 Prognosequalität einer Umfrage

Sechs zufällige Personen aus allen Stimmberechtigten

Wenn wir nun die Personen A-F befragen würden und die Umfrage-Ergebnisse veröffentlichen, dann würde das Ergebnis so aussehen: JA 50% zu NEIN 50%. Das wäre ein Fehler von 17%. Das Problem bei dieser Umfrage ist, dass sie nicht im Geringsten repräsentativ ist. Wenn wir die Personen A-F völlig zufällig aus der Testgruppe herausgezogen haben, dann haben wir zwar eine hohe Wahrscheinlichkeit, dass das Bild etwa der realen Verhältnisse entspricht, haben aber auch eine gewisse Wahrscheinlichkeit, dass wir eine Stichprobe haben, die komplett anders aussieht. Wenn wir also absolut zufällig irgendwelche Leute befragen, können wir nicht abschätzen, wie repräsentativ diese Umfrage tatsächlich ist. Eine völlig zufällige Stichprobe ist für eine qualitativ hohe Umfrage nur dann geeignet, wenn eine sehr hohe Zahl von Personen befragt wird, die verursacht aber zu hohen Kosten.

Es kann aber auch sein, dass die Stichprobe nicht zufällig ist, sondern z.B. nur aus Männern besteht oder nur aus 20- bis 30-Jährigen. Dann ist die Wahrscheinlichkeit hoch, dass das Bild nicht mehr der realen Verhältnisse entspricht. (Bei den Benutzern einer Wahlbörse ist dies sicher der Fall, da nur Personen mit Internetzugang teilnehmen können, was bereits eine bestimme Auswahl aus der Zielgruppe bedeutet.)



Zufällig ausgewählte Stichproben

Wir haben also zwei Dilemmas: Das erste ist, dass eine völlige zufällig ausgewählte Gruppe nur zu einer bestimmten Wahrscheinlichkeit ein reales Bild liefert. Die zweite ist, wie wir es überhaupt anstellen, eine völlig zufällig ausgewählte Gruppe zu haben.

Bei repräsentativen Umfragen kann man dies nun wie folgt lösen: Man untersucht die Zielgruppe (z.B. den Souverän und zwar nur die tatsächlich stimmende Bevölkerung und nicht die stimmberechtigte) und bildet Kategorien. Die genauste Umfrage hätte man, wenn man für jede Person eine eigene Kategorie bildet. Die Umfrage wäre dann aber am schwersten, denn man müsste pro Kategorie auch wieder eine Person befragen. Die Umfrage ist somit absolut äquivalent zur Abstimmung am Abstimmungssonntag. Die Idee ist, dass man also Kategorien wählt, wo die Meinung am ehesten konstant bleibt. Zwei Personen in der gleichen Lebenssituation haben viel wahrscheinlicher eine gleiche Meinung als zwei Personen in völlig unterschiedlichen Lebenssituationen. Es ist deshalb von Vorteil möglichst Kategorien zu finden, die passend eine solche Lebenssituation beschreiben. Attribute wären da z.B. Geschlecht, Alter, Lohn, Beruf, Ausbildung, Sprachregion, etc.

Kategorien:
Nehmen wir an, dies sei eine mögliche Verteilung nach Kategorien. Die Wahrscheinlichkeit, dass Personen in diesen Kategorien dieselbe Meinung haben ist bereits sehr hoch. Wenn wir nun noch

	ohne Lohn		- Fr. 25'000		- Fr. 75'000		Fr. 75'000 +	
	Mann	Frau	Mann	Frau	Mann	Frau	Mann	Frau
18-20	5.1%	2.1%	2.0%	7.0%	3.0%	1.1%	0.9%	2.5%
21-30	1.2%	7.2%	2.2%	1.1%	1.0%	2.2%	0.8%	1.5%
31-40	3.2%	3.2%	7.7%	1.2%	1.0%	1.0%	2.0%	0.5%
41-50	3.9%	1.3%	2.3%	2.1%	2.1%	1.1%	1.5%	3.1%
51+	2%	3.5%	0.5%	0.7%	3.1%	7.1%	2.5%	2.5%

zufällig Personen aus den Kategorien auswählen haben wir nochmals eine zusätzliche hohe Wahrscheinlichkeit, dass ihre Meinung für die Kategorie repräsentativ ist. Wir wissen nun auch, wie wir die Meinung einer Person im Gesamtkontext gewichten müssen. Die Meinung eines arbeitslosen Mannes mit Alter 25 ist zu 1.2% zu gewichten im Gegensatz zu den 0.5%einer sehr gut verdienenden Frau mit Alter 35. Wenn wir also für jede Kategorie Personen finden und die Meinung dieser Personen gewichten, dann können wir sehr gut abschätzen, wie gut unsere Umfrage zur realen Situation ist. Und die Qualität bleibt bei jeder Umfrage konstant hoch.

3.6.2. Prognosequalität einer Prognosebörse

Wie wir also gesehen haben ist bei repräsentativen Umfragen die Auswahl der Personen enorm wichtig.

Die Frage ist nun, ob es eine Möglichkeit gibt, die Qualität einer Prognose unabhängig von der Auswahl der Personen zu machen. Die Lösung dabei ist, dass man nicht die persönliche Meinung, sondern die persönliche Einschätzung der Personen zu Rate zieht. Wenn wir nochmals die Personen A-F nehmen (welche für die Zielgruppe nicht im geringsten repräsentativ sind) und fragen, wie denn die durchschnittliche Einschätzung aussieht, dann zeigt sich das folgende Bild:

JA (75%+70%+65%+65%+60%+55%) : 6 Personen = 65% (real 67%)
NEIN (25%+30%+35%+35%+40%+45%) : 6 Personen = 35% (real 33%)

Diese Einschätzung kommt nun bereits sehr nahe an die Realität heran.

Unsere Prognosebörse muss aber zwei Anforderungen erfüllen, um die Prognosequalität zusätzlich zu steigern:

1. Diejenigen, die am besten die Situation einschätzen, werden mit einem finanziellen Gewinn belohnt. Diejenigen, die sich verschätzen, müssen einen finanziellen Verlust hinnehmen. Das soll dazu verleiten, sich möglichst genau ein Bild vom tatsächlichen Ausgang zu machen. Zweitens liefert das Portemonnaie eine gute Information, ob ein Händler nicht doch seine Einschätzung über den Abstimmungsausgang korrigieren muss. (Ob es sich dabei um reales Geld handeln muss, ist fraglich. Evtl. ist der optimale Weg, wenn ein Händler einen minimalen echten Betrag einsetzen muss, weil das bereits psychologisch wirkt.)

2. Über die Handelsplattform soll ein einfacher Zugang zu den Informationen gewährleistet werden, die eine Einschätzung über einen Abstimmungs- bzw. Wahlausgang beeinflussen. Das bedeutet also, dass Informationen über Ergebnisse ähnlicher Abstimmungen oder auch Ergebnisse aus Umfragen und auch Nachrichten über die Plattform erhältlich sein müssen. Genau dies ermöglicht eine fortwährende Reflexion der eigenen Einschätzung über das Handelsgeschehen und verfeinert so zusätzlich die Qualität der Prognose.

Das Ziel ist also eine möglichst gute Prognose zu erhalten. Deshalb ist bei der Realisation gerade auch Punkt 2 wichtig. Natürlich kann man argumentieren, dass doch bereits Punkt 1 dazu führt,

dass die Prognosequalität da ist. Wie ich aber bei früheren Wahlbörsen (Nationalratswahlen 1999 und 2003) feststellen musste, ist dies nicht automatisch der Fall. Wenn eine Mehrheit der Händler sich verschätzt, werden sie erst beim Abstimmungs- und Wahlsonntag finanziell bestraft. Diejenigen, die richtig eingeschätzt haben, werden erst dann belohnt. Z.B. waren „Andere"-Aktien bei der Wahlbörse von 1999 und 2003 chronisch unterbewertet. Da sich aber die Mehrheit nicht um die Ergebnisse der vorhergehenden Legislatur gekümmert hat, hat sich der Kurs auch nie nach oben korrigiert. Obwohl ich schlussendlich als Händler zu den Gewinnern gehörte, war die Qualität der Prognose bis zum Wahlsonntag schlecht. Die Prognosequalität ist aber das Ziel der Börse und deshalb muss hier bereits entgegengewirkt werden.

Ein zweites Argument gegen Punkt 2 ist, dass es doch ein Widerspruch ist, Umfragen auf der Plattform zu veröffentlichen. Eine Umfrage und eine Wahlbörse sind doch erstmals sehr gegensätzliche Prognoseinstrumente. Ich bin aber der Meinung, dass dies nicht der Fall sein muss. Eine Prognosebörse kann gerade dazu führen den Wahrheitsgehalt einer Umfrage nochmals zu bewerten. Somit ist eine Umfrage eine gute Orientierung für die Händler. Und machen schlussendlich die Prognose besser.

4 Projektmanagement

4.1. Prozess-Modell

Die Idee zu dieser Diplomarbeit kam von unserer Seite. Ausgehend von den Arbeiten aus der zweiten Projektarbeit (PA2) im Sommersemester, haben wir diese Wahl- bzw. Abstimmungsbörse implementiert. Ziel bei der PA2 war es in erster Linie eine Börsensimulation zu implementieren. Obwohl sich eine Börse im Zweck von einer Wahlbörse unterscheidet, sind beide im Kern sehr ähnlich. Wir sind also mit den Resultaten aus der PA2 gestartet.

Für den Softwareentwicklungsprozess haben wir zu Beginn ein eigenes Prozess-Modell entwickelt. Dafür haben wir uns an bekannten Prozessmodellen orientiert.

Unser Modell ist ein Mischung zwischen des Evolutionären/inkrementellen Modells, Rational Unified Process (RUP) und Extreme Programming (XP). Von RUP stammen die Disziplinen, die wir in jeder Iteration durchgeführt haben. Das waren: Anforderungen, Analyse, Design, Implementierung und Test. Im Gegensatz zu RUP bearbeiteten wir alle Disziplinen in jeder Iteration gleich stark. Vom Evolutionären/inkrementellen Modell stammt, dass nach jeder Iteration eine lauffähige Version vorhanden war. Aus XP liessen wir einige Praktiken, wie das Pair-Programming und Simple-Design in unser Prozessmodell einfliessen.

Typischerweise sah eine Woche so aus:

Montag:	Anforderung + Analyse
Dienstag:	Design
Mittwoch:	Implementierung
Donnerstag:	Implementierung
Freitag:	Test

Die Definition des Softwareentwicklungsprozesses hat nicht als Zweck uns in unserem Tun einzuengen, sondern uns einen Rahmen zu geben, an dem wir uns ausrichten können. Jeden Freitag haben wir die kommende Iteration kurz besprochen. So haben wir beispielsweise vor der dritten Iteration erkannt, dass die nächste Iteration über Statistik und Charts zu früh geplant war. Wir haben uns entschieden ASP.NET und der Datenbankentwurf den Statistiken vorzuziehen. Für die Implementierung der Webseite, war meist keine Analyse mehr nötig, weil wir für die Applikation bereits voll spezifizierte Anwendungsfälle geschrieben hatten und diese genauso für die Webseite galten.

Zirka eine Woche vor dem geplanten Zeitpunkt haben wir den Testbetrieb “live“ geschaltet und haben festgestellt, dass das System bisher fehlerfrei läuft. Bis zum heutigen Zeitpunkt haben sich rund 30 Händler registriert. Wir hoffen, dass sich bis zur Präsentation noch einige mehr anmelden, damit die Prognosebörse ihren Zweck erfüllen kann.

4.2 Verlauf

4.2.1 Erste Iteration

Aus PA2 stammt das Wissen wie eine Börse funktioniert und das Wissen wie man ein Auftragsbuch implementiert. Auf die neuen Anforderungen angepasst, haben wir die Börse analysiert und designt. Wir sind in der erste Woche (=erste Iteration) in der Entwicklung der Business-Logik weiter gekommen, als während der PA2 in einem ganzen Semester. Wir konnten uns während der DA intensiv und exklusiv mit dem Thema befassen, was während der PA2 nicht möglich war. Wir haben die Use-Cases für das Handeln geschrieben.

Am Ende der ersten Iteration haben wir noch den Test-Server hardwaremässig aufgesetzt. Die Installation der Datenbank lief erfolglos.

4.2.2 Zweite Iteration

In dieser Iteration haben wir unsere Business-Logik mit der Funktionalität des Depots erweitert. Das heisst, dass neben dem Handel im Auftragsbuch jetzt auch gleich ein Depot nachgeführt wird für jeden Benutzer mit den dazugehörigen Aktien.

In dieser Woche haben wir beim Design entschieden, ein offenes System einem restriktiven vorzuziehen. (Eine Diskussion über beide Systeme finden Sie im Kapitel 7.3) Dieser Entscheid bringt dem Händler viel Freiheit beim Handeln, ist aber einiges komplexer in der Entwicklung. Was vor allem Arbeit machte, war die Entwicklung der rekursiven Transaktion.

Am Ende hatten wir eine lauffähige Version.

Wir haben am Ende dieser Iteration den MS-SQL-Server aufgesetzt.

4.2.3 Dritte Iteration

In dieser Iteration drängte sich ein Code-Review auf, weil die ganze Transaktionssteuerung noch in der Bank-Klasse lief.

Wir haben über Last-Tests Mitte der Woche festgestellt, dass die Applikation verschwenderisch mit dem Speicher umgeht. Es wurden alle Instanzen der Auftragsbücher und Depots ständig im Speicher gehalten. Um diesem Problem entgegenzuwirken haben wir erste Versuche gemacht, den Speicher mit Hilfe eines LRU (english: Least-Recently-Used) Mechanismus zu verwalten.

In dieser Woche wurde ASP.NET zum ersten Mal in Angriff genommen. Wir betrachteten ASP.NET als ein grosses Risiko und wollten es möglichst früh in den Griff bekommen. Resultat der ersten ASP.NET-Gehversuche war eine Login-Maske, wo man sich über eine Datenbanktabelle mit Username und Passwort authentisieren konnte. Zudem hatten wir mit Hilfe der Session bereits die Möglichkeit nach der Authentisierung eine persönliche Webseite mit Daten aus der Datenbank anzuzeigen.

4.2.4 Vierte Iteration

Um dem Problem des überbordenden Speichers Herr zu werden, prüften wir einen anderer Ansatz mit “Weak References“. Diese genügten unseren Anforderungen aber nicht. Die Lösung war schlussenldich eine prioritätsgesteuerte Linked-List, die nach jeder Transaktion zurecht geschnitten wird. (Realisiert in der Klasse ObjectManager.)

Da wir grössere Fehler in DataViews feststellten, haben wir den ganzen Kern neu programmiert.

Wir haben in dieser Iteration Beans entworden (inspiriert von Entity-Beans von J2EE). Ein Bean ist ein Tupel aus der Datenbank, abgebildet auf eine einfache Datenstruktur. Mit der

Einführung der Beans wurde die Datenverarbeitung mit der Datenbank stark vereinfacht und auf unsere Bedürfnisse angepasst.

Wir haben die Transaktion in unserem System nochmals genau untersucht. Die Einführung einer Regel, dass Händler nie mit sich selber handeln dürfen, hat unser System zusätzlich vereinfacht. Ab sofort mussten wir für die Rekursion nur noch Kaufaufträge berücksichtigen und nicht wie bis anhin auch Verkaufaufträge.

Am Ende dieser Iteration haben wir den Kern gründlich mit Blackboxtests getestet.

4.2.5 Fünfte Iteration

Zu Begin waren noch geringfügige Anpassungen an die Businesslogik erforderlich.

Dann entwickelten wir ASP.NET-GUIs. Aus den Anforderungen aus der ersten Iteration hatten wir schon einen Dummy-Ansicht, wie unsere Webseite auszusehen hat. Unsere Aufgabe bestand also darin, diese Dummy-Ansicht in echte ASP.NET-Webseiten umzuschreiben.

Wir haben zu Beginn mit vorgefertigten Repeater-Controls gearbeitet. Dies endete in einer Sackgasse, da wir mit vorgefertigten Controls niemals unser komplexes System abbilden konnten.

Die Entscheidung, dass wir eigene Webcontrols schreiben müssen, fiel Ende dieser Iteration.

4.2.6 Sechste Iteration

Mit den Webcontrols konnten wir unsere Probleme aus der vorangehenden Iteration lösen. Wir haben alle Webcontrol implementiert und alles zu einem Webauftritt verdichtet.

Die Usability der Webseite liess noch zu wünschen übrig, was wir in dieser und der nächsten Iteration verbesserten. Hier war noch unklar ob und wann wir den Testbetrieb aufschalten konnten.

4.2.7 Siebte Iteration

In dieser Iteration haben wir nach Anpassungen auf der Webseite und Härtung des Serversystems, den Testbetrieb des Politmarket live geschaltet. Der eingesetzte Server ist ein Windowsserver und vereinigt Business-Logik-Tier und Datenbank-Tier und wird während dem gesamten Testbetrieb privat bei Herrn Rieser betrieben.

Wir haben Kollegen, Freunde und Bekannte aufgefordert die Webseite zu testen. Der Ansturm verhielt sich allerdings in Grenzen. Einige Benutzer fühlten sich überfordert mit der Komplexität des Systems, einige hatten keine Zeit und wiederum andere wurden erschlagen vom vielen Text auf der Start- und Spielregeln-Seite.

Wir können trotzdem sehr zufrieren zurückblicken auf diesen Moment, weil sich das System zum ersten Mal als funktionsfähig und stabil erwies.

Die zweite Hälfte dieser Iteration widmeten wir bereits der Vervollständigung der Dokumentation.

4.2.8 Achte Iteration

In der letzten Iteration haben wir die Dokumentation zusammengestellt und abgeschlossen. Neben der Dokumentation haben wir den Testbetrieb beobachtet und uns bereits Ausbauschritte überlegt.

4.2.9 Fazit

Das Ziel eine funktionierende Wahlbörse zu implementieren haben wir erreicht. Dass wir sogar erfolgreich das Resultat live schalten konnten und richtige Benutzer auf der Plattform handeln können, bestätigt uns in unserer Vorgehensweise und der Qualität unserer Arbeit.

Währen dem Testbetrieb haben wir bereits einige interessante Feedbacks von Benutzern erhalten. Leider war es uns in dieser kurzen Zeit nicht mehr möglich Anpassungen vorzunehmen. Sollten wir aber das Projekt nach der Diplomarbeit weiter betreiben, werden wir diese Anpassungen vornehmen.

- Editierbarkeit der Aufträge in der Depotübersicht
- Kaufpreisberechnung in der Depotübersicht
- Nachführung von Portfoliokäufen und -verkäufen

Weiter haben wir schon Ausbauschritte überlegt, die auch erst später implementiert werden können:

- Administrationstool: für eine einfache Verwaltung des Handels, für das Monitoring des gesamten Systems und der Benutzerverwaltung
- Content Management: für eine einfache und schnelle Aufschaltung neuester Nachrichten
- automatisiertes Backup der Datenbank: um den Verlust von Daten zu verhindern
- Redundanter Datenbank-Server: damit bei Ausfall weiter gehandelt werden kann.
- Ausbau der Statistikfunktion auf der Webseite: für individuellere Studien

Wir rechnen damit den Polimarket nach der Diplomarbeit weiter zu betreiben. Wir haben von Anfang an diese Prognosebörse aus eigenem Interesse implementiert und deshalb sind wir sehr interessiert daran, dieses Projekt weiter zu betreiben. Es liegt nicht in unserem Interesse, das Produkt gewinnorientiert zu betreiben, sondern lediglich als ein persönliches Hobby zu betreiben.

5 Analyse

5.1 FURPS+

5.1.1 Functional Requirements

Der Benutzer muss in Folgendes Einsicht haben: Anzahl Aktien, Anzahl Geld, hängige Aufträge, Gewinn/Verlust, Kontobewegungen.
Der Benutzer muss Aufträge eingeben und löschen können.
Folgende allgemeine Informationen sind für den Benutzer zugänglich: Statistiken über den Kursverlauf der Aktien über Stunden und Tage, Ausschnitt aus den Auftragsbüchern (Bid/Ask), Handelsschluss, Letzter Kurs, Ergebnisse aus Umfragen und vergangenen Abstimmungen, die Spielregeln.

5.1.2 Usability

Die Börse muss über Web-GUIs erreichbar sein. Unterstützt werden neuste Versionen von Firefox und Internet Explorer. Die Seiten müssen klar, übersichtlich und in gewissem Masse intuitiv erfassbar sein.

5.1.3 Reliability

Das System muss 24 Stunden zugänglich sein, mit wenigen Support-Ausfällen.

5.1.4 Performance

Das System muss mit 2000 Benutzern klarkommen. Dabei muss es auch bei zehn concurrent Usern noch eine ansprechende Geschwindigkeit haben. (Reaktionszeit bei neuen Aufträgen $t < 1$ sec.).

5.1.5 Supportability

Die Web-GUIs müssen einfach administrierbar sein um neue Umfragen und neue News aufzuschalten. Dazu sollten einfache Webprogrammierkenntnisse ausreichen.

5.1.6 + Security

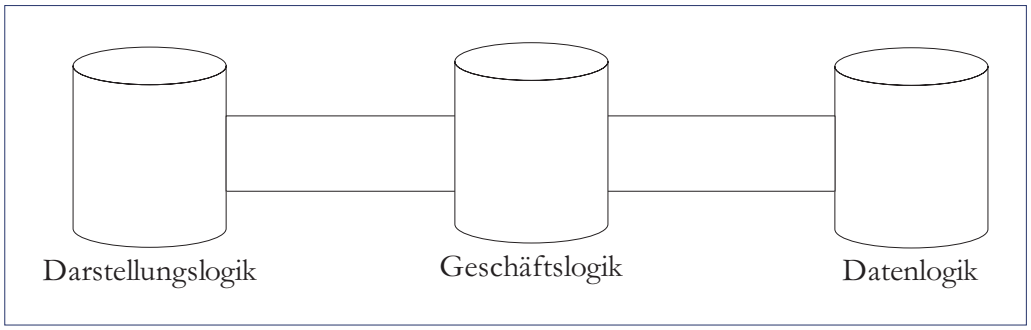
Das System hat eine adäquate Sicherheit. Die Benutzer müssen identifiziert werden und die Accounts sind mit Passwort geschützt. Aufträge aufzugeben und zu löschen müssen in Transaktionen ablaufen. Die Architektur muss so gestaltet sein, dass zusätzliche Sicherheitsvorkehrungen, wie HTTPS und Firewall in verschiedenen Ebenen ermöglicht wird.

5.1.7 + Independence

Das System muss so designet sein, dass es nicht komplett von einer Technologie oder Plattform abhängig ist. Es müssen also verschiedene Datenbank-Systeme und verschiedene Webserver-Systeme unterstützt werden.

5.2 Die 3-Tier-Architektur

Das von uns gewählte Softwarekonzept ist wie in der PA2 die 3-Tier-Architektur (deutsch: Drei-Schicht-Architektur). Eine 3-Tier-Architektur beschreibt eine Client-/Server-Architektur, in der die Darstellungslogik, die Geschäftslogik und die Datenlogik voneinander getrennt werden. Meist werden die drei Schichten nicht nur konzeptionell getrennt, sondern gleich auf separaten Systemen gehalten.



Bildlegende

Ziel ist es dabei, die drei Schichten unabhängig voneinander zu betreiben und so die Vorteile eines modularen Aufbaus der Software zu haben. Die Schichten kommunizieren dabei über klar definierte und kontrollierbare Schnittstellen. Das führt dazu, dass eine Änderung in einer Schicht eine weitere Schicht soweit in Mitleidenschaft zieht wie es die Schnittstelle erfordert.

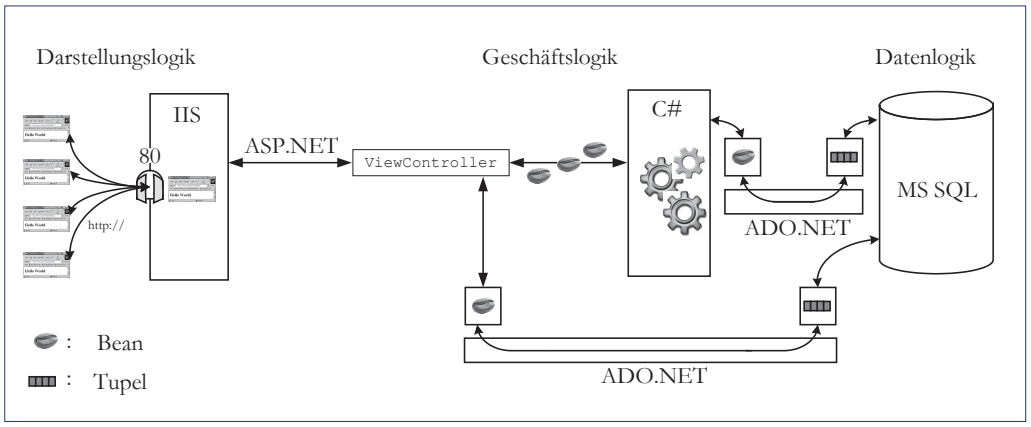
Ein Beispiel: Würden wir die Datenlogik von MS SQL auf ein anderes SQL-kompatibles DBMS wie beispielsweise Oracle ändern, so müssen wir lediglich die Verbindung neu definieren.

Typischerweise wird die Darstellungslogik von einem Webserver repräsentiert, der von einem beliebigen PC mit Browser über http (Hyper Text Transfer Protokoll) erreicht werden kann. Die Darstellungsschicht ist also sowohl server- wie auch clientseitig.

Die Geschäftslogik wird meist schon, je nach Performance, auf einem dedizierten Applikationsserver betrieben. Die Datenlogik besteht meistens aus einem Datenbank-Management-System auf einem eigenen Rechner.

Es ist auch denkbar, dass die Geschäftslogik auf mehrere Applikationsserver verteilt wird. In diesem Fall spricht man von einer n-Tier-Architektur.

Übertragen auf unsere Diplomarbeit präsentiert sich die Abbildung X jetzt so:



Es bleibt uns noch die Schnittstellen für die Kommunikation zwischen den Schichten zu definieren.

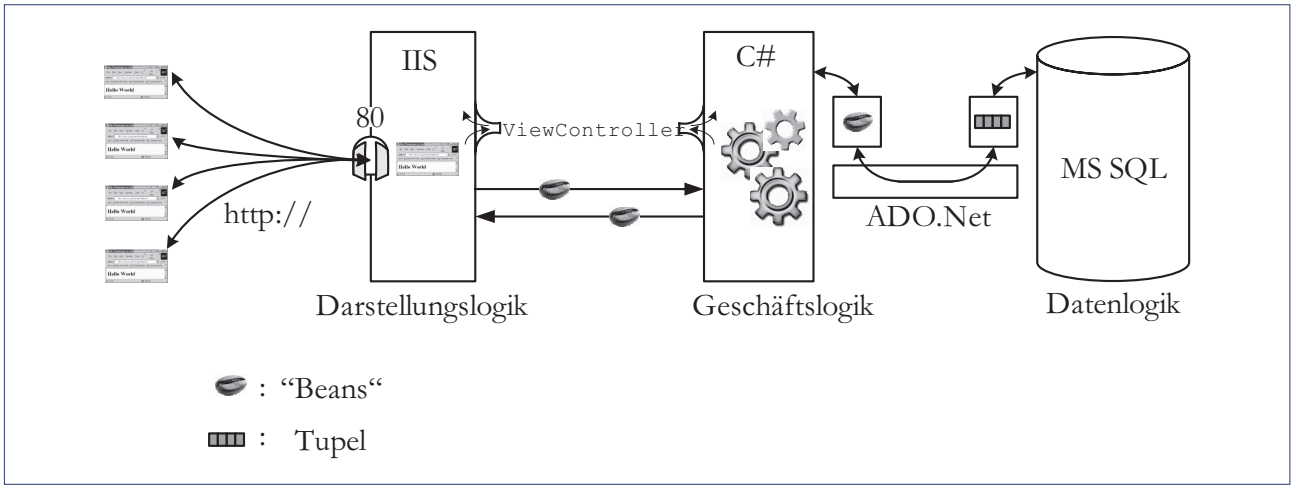
Die Kommunikation zwischen Darstellungslogik und der Geschäftslogik findet auf zwei Ebenen statt: Auf der einen Seite haben wir die Klasse ViewController, die Anfragen aus der Darstellungslogik entgegennimmt und koordiniert an die Geschäftslogik weiterleitet. Auf der

anderen Seite haben wir die Beans, die eine Abbildung eines Tupel in einer Datenstruktur ist. Das Bean wird dann während einer Transaktion von der Geschäftslogik geändert, gelöscht oder neu generiert. Gibt man nun einen Auftrag ein, wird dieser vom ViewController entgegengenommen und an die Geschäftslogik zur Verarbeitung weitergeleitet. Wird allerdings die Depotübersicht auf der Webseite aufgebaut, holt sich die Darstellungslogik mit Hilfe der Beans die Daten direkt aus der Datenbank.

Die Schnittstelle zwischen der Geschäfts-Logik und den Daten wird uns bereits vom .NET Framework angeboten. Das API (englisch: Application Programming Interface) trägt den Namen ADO.NET und ist ein Neuentwicklung vom früheren ADO (englisch: ActiveX Data Object). ADO.NET bietet uns einerseits einen verbindungsorientierten- wie auch verbindungslosen Zugriff auf die Datenbank. Der verbindungsorientierte Datenbankzugriff öffnet direkt eine Verbindung auf die Datenbank, bearbeitet die Daten und schliesst am Ende die Verbindung wieder. Eine Datenbank kann mit mehreren offenen Verbindungen umgehen. Wir haben aber für die Transaktion in der Geschäftslogik eine hohe Isolationsstufe gewählt, um die Datenkonsistenz auf der Datenbank zu gewährleisten. Das heisst, dass wir während einer Transaktion nur eine Verbindung zulassen.

Wir haben zu Beginn der DA den verbindungslosen Zugriff eingesetzt, mussten aber feststellen, dass in einem transaktionsorientierten System, wie wir es aufgebaut haben, der verbindungslose Zugriff keine Vorteile bringt. Im Gegenteil war der Umgang mit den DataSets und DataViews umständlicher.

Verfeinern wir nun das Schema mit den soeben gegebenen Zusatzinformationen:



Bildlegende

5.3. Restriktives vs. offenes System

Wir stellten die Anforderungen, dass das Geld in einem Depot nie unter 0 fallen und ein Händler keine Leerverkäufe tätigen darf (Verkäufe von Aktien, die er nicht hat). Diese Anforderung stellte uns aber schon sehr früh an eine Entscheidung: Entweder machen wir ein sehr restriktives oder ein sehr offenes System.

5.3.1. Restriktives System

Das System überprüft bei einem Kaufauftrag, wie viel Geld ein Benutzer noch hat und wie viele Kaufaufträge er bereits im System hängig hat. Hat er zuwenig Geld für den Kaufauftrag, verweigert das System die Annahme des Auftrags. Ebenso würde es bei Verkaufsaufträgen funktionieren: Das System überprüft zuerst, ob der Benutzer die Aktien besitzt und ob er nicht schon zu viele Verkaufstaufträge von diesen Aktien im System hat. Wenn alles in Ordnung ist, würde das System den Auftrag annehmen, sonst aber ablehnen.

Ein solches System würde die Anforderungen problemlos erfüllen. Ebenso hätte es den Vorteil, dass es relativ einfach zu realisieren ist.

Der Nachteil dabei ist, dass es dem Benutzer sehr viele Freiheiten wegnimmt. Es gibt da zum Beispiel verschiedene Händler-Strategien:

a) Eine Strategie ist, dass der Benutzer von zwei verschiedenen Aktien Kaufaufträge eingibt, welche noch unter Kurs sind. Er möchte dann einfach den Auftrag ausgelöst haben, dessen Kurs zuerst fällt.
b) Oder er möchte ein Kauf- und Verkaufsauftrag einer Aktie eingeben und hofft, dass der Kurs zuerst fällt, so dass er sie kaufen kann. Er hat dann sofort einen Verkaufsauftrag im Auftragsbuch, um die Aktien wieder gewinnbringend zu verkaufen.

All diese Strategien sind bei einem restriktiven System aber nicht möglich. Bei a) würde das System die Annahme verweigern, wenn beide zusammen das maximal verfügbare Geld überschreiten würden, obwohl jeder Auftrag für sich alleine ausführbar wäre. Bei b) würde das System die Annahme verweigern, wenn der Benutzer nicht die nötigen Anzahl Aktien hätte, obwohl er sie evtl. über den Kaufauftrag zukaufen würde, bevor der Verkaufsauftrag ausgelöst wird.

5.3.2. Offenes System

Das offene System würde alle Aufträge grundsätzlich einmal annehmen. Das Problem ist, dass man so die Anforderungen, dass das Geld im Depot nicht unter 0 fallen darf oder die Aktien vorhanden sein müssen, nur sehr schwer erfüllbar sind. Wenn man jetzt Aufträge annimmt und diese in ein Auftragsbuch schreibt, dann stehen dort also Aufträge, die bei einem Match gegen unsere Anforderungen verstossen würden. Das Auftragsbuch müsste also vor jedem Match überprüfen, ob es diesen Match überhaupt durchführen darf. Das Regelwerk dazu würde enorm komplex ausfallen. Sehr unschön dabei ist, dass jemand, der Einblick in das Auftragsbuch erhält (und wir wollen dem Benutzer ein wenig Einblick in das Auftragsbuch gewähren), nie wissen kann, ob diese Volumen nun echt handelbar sind oder schlussendlich doch vom System ignoriert werden müssen.

5.3.3 Restriktiver Kern in offenem System

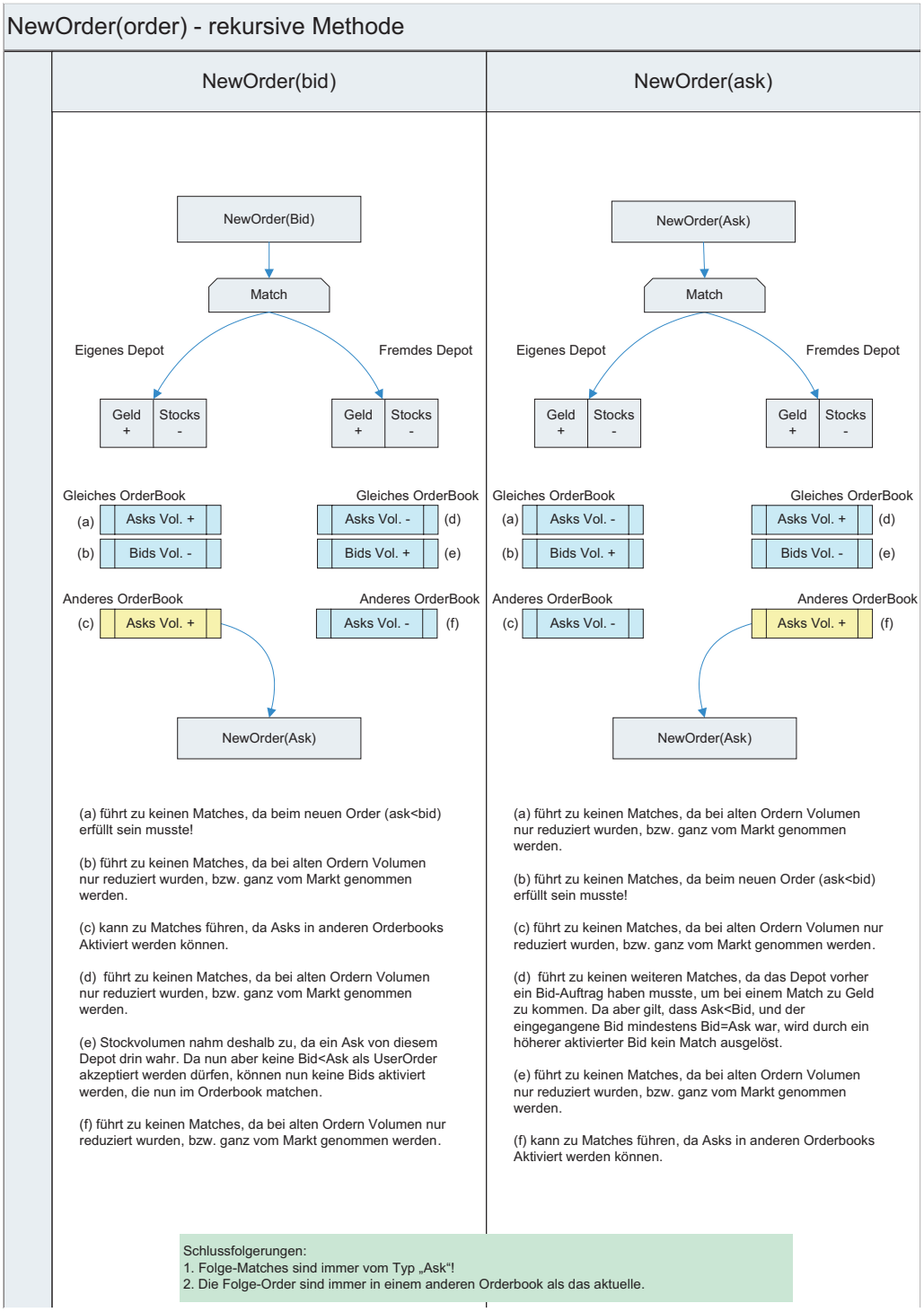
Weil wir dem Benutzer nicht den Spass verderben wollten, haben wir uns für ein offenes System entschieden. Aber wir haben einen restriktiven Kern eingebaut. Von aussen sieht es für den Benutzer aus, als wären alle Aufträge erlaubt. Das einzige, was er erhält, sind Warnungen. In den Auftragsbüchern werden aber nur die Aufträge nachgeführt, die im Moment zu 100% ausführbar sind. (Wenn im Auftragsbuch steht, man kann 500 Aktien zu 12 kaufen, dann ist das auch so.) Dafür mussten wir das Konzept von zwei verschiedenen Aufträgen einführen. Es gibt Aufträge des Benutzers (UserOrder), die absolut nicht restriktiv sind und unseren Anforderungen widersprechen können, und Aufträge für die Auftragsbüchern (SystemOrder), die restriktiv und nur so dimensioniert sind, dass sie unseren Anforderung nicht widersprechen können.

Wenn nun ein neuer Auftrag vom Benutzer eingegeben wird, dann muss daraus ein SystemOrder erzeugt werden, der ins Auftragsbuch eingetragen werden kann. Dieser Auftrag kann nun einen Handel auslösen. Einen Handel auslösen heisst, dass ein Benutzer nun mehr Aktien und weniger Geld besitzt und ein anderer Benutzer nun weniger Aktien und mehr Geld. Das Problem ist nun, dass ein Handel die Grundlage eines Benutzers ändert und so ändert sich auch die Grundlage für die SystemOrders. Z.B. hatte ein Benutzer vor dem Handel einen UserOrder im Depot um von einer Sorte Aktien 500 Stk. zu kaufen. Da der Benutzer aber kein Geld besass, wurde im restriktiven Kern kein SystemOrder eingetragen. Nach dem Handel könnte dieser Benutzer aber nun Geld besitzen und tatsächlich diese 500 Stk. kaufen. Es muss also nach jedem Handel überprüft werden, ob die SystemOrders bei den beteiligten Parteien noch korrekt ist und neu generiert werden. Das Ganze hat zusätzlich einen rekursiven Charakter: Ein aktivierter Kaufauftrag kann zu einem Handel führen, wo ein Dritter zu Geld kommt, wo wiederum ein Kaufauftrag aktiv wird, und ein Vierter zu Geld kommt, etc. Das Ganze wird solange fortgesetzt, bis ein Zustand erreicht ist, wo kein Auftrag mehr zu einem neuen führt. Da

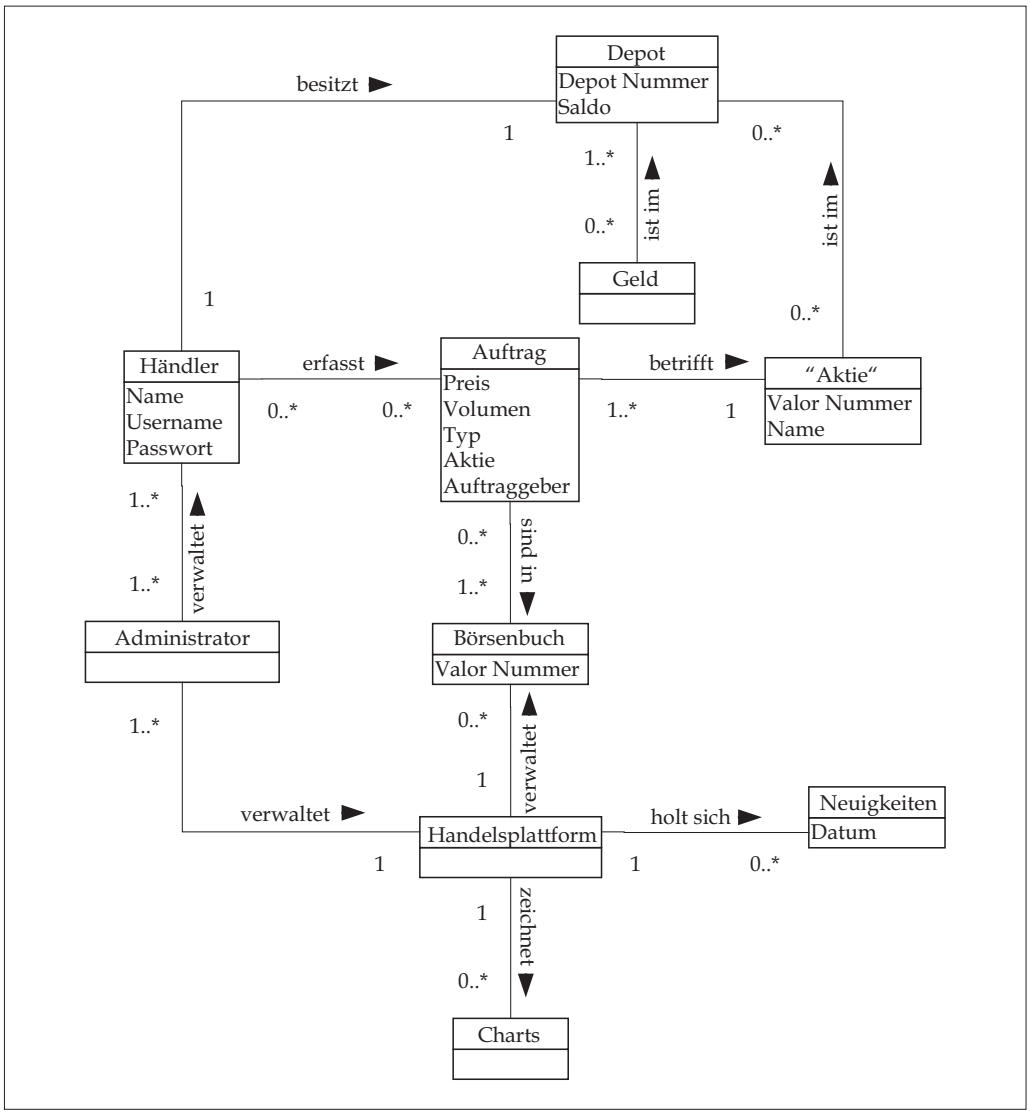
das System eine endliche Anzahl Aufträge enthält, ist dieser rekursive Durchgang auch endlich. Es kann nicht sein, dass sich eine unendliche Schleife bildet, selbst dann nicht, wenn sich ein Benutzer die Aktien selber zuschanzt.

Unsere Analyse hat gegeben, dass wir das ganze System aber zusätzlich vereinfachen können, wenn wir genau dies wiederum verhindern. Ein Benutzer darf sich selber keine Aktien verkaufen. Kaufaufträge und Verkaufsaufträge, die ein Benutzer eingibt, dürfen sich auf keine Fälle gegenseitig matchen. Diese Restriktion hielten wir für sinnvoll.

5.3.4 Analyse des restriktiven Kerns im offenen System



5.4 Das Domänen Modell



Händler:
Jeder Händler besitzt ein Depot und kann mittels Aufträgen Aktien kaufen und verkaufen. Der Händler wird von einem registrierten Benutzer gesteuert.

Depot:
Im Depot sind nicht nur die Aktien des Händlers vorhanden, sondern auch die flüssigen Mittel.

Auftrag:
Aufträge betreffen Aktien und werden in das entsprechende Börsenbuch abgelegt. In einem Auftrag stehen neben Preis und Volumen, auch der Typ (kaufen bzw. verkaufen), die betreffende Aktie und der Auftraggeber.

Börsenbuch:
In einem Börsenbuch werden jeweils alle hängigen Aufträge einer Aktie gehandelt. Die Aufträge werden nach den bekannten Regeln im Börsenbuch gemachet.

Aktie:
Die Aktie hat eine Valorennummer zu Identifikation und einen Namen. Der aktuelle Preis wird im Börsenbuch ermittelt.

Handelsplattform:
Die Handelsplattform verwaltet alle Börsenbücher. Aus den Informationen der Handelplattform lassen sich Charts generieren.

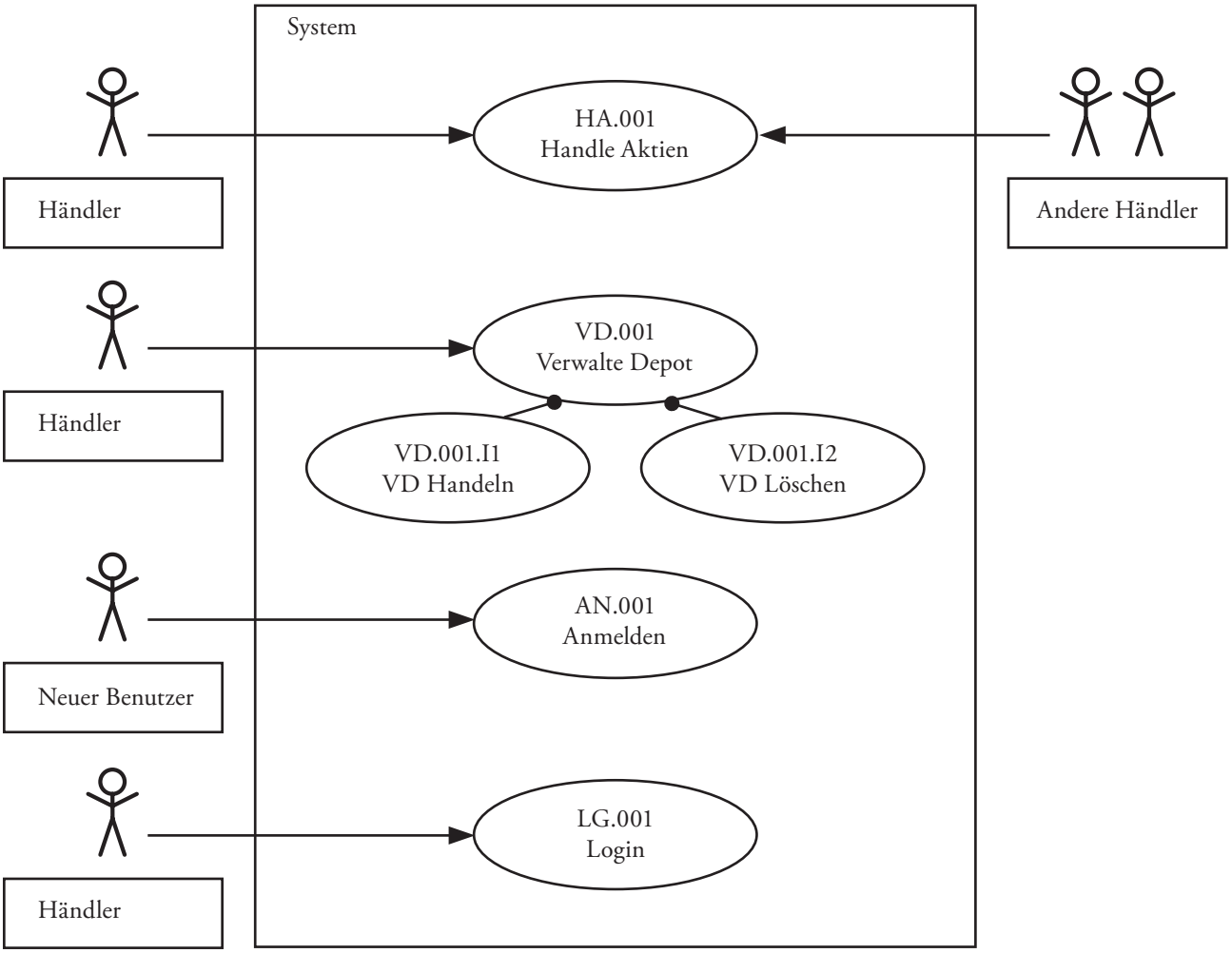
Administrator:
Der Administrator verwaltet alle Händler und über die Handelsplattform das Geschehen am Markt. Der Administrator kann zum Beispiel Händler sperren oder neue Börsenbücher für neue Aktien hinzufügen.

5.5 Anwendungsfälle

5.5.1 Anwendungsfalldiagramm

Folgende Anwendungsfälle sind beschrieben worden:

Handle Aktien	Händler kauft bzw. verkauft Aktien	“fully-dressed“
Verwalte Depot	Händler löst bzw. löscht Aufträge aus der Depotübersicht aus.	“fully-dressed“
Anmelden	Neuer Benutzer meldet sich an.	“casual“
Login	Händler loggt sich ein.	“brief“



5.5.2 Handle Aktien (HA.001)

Use case

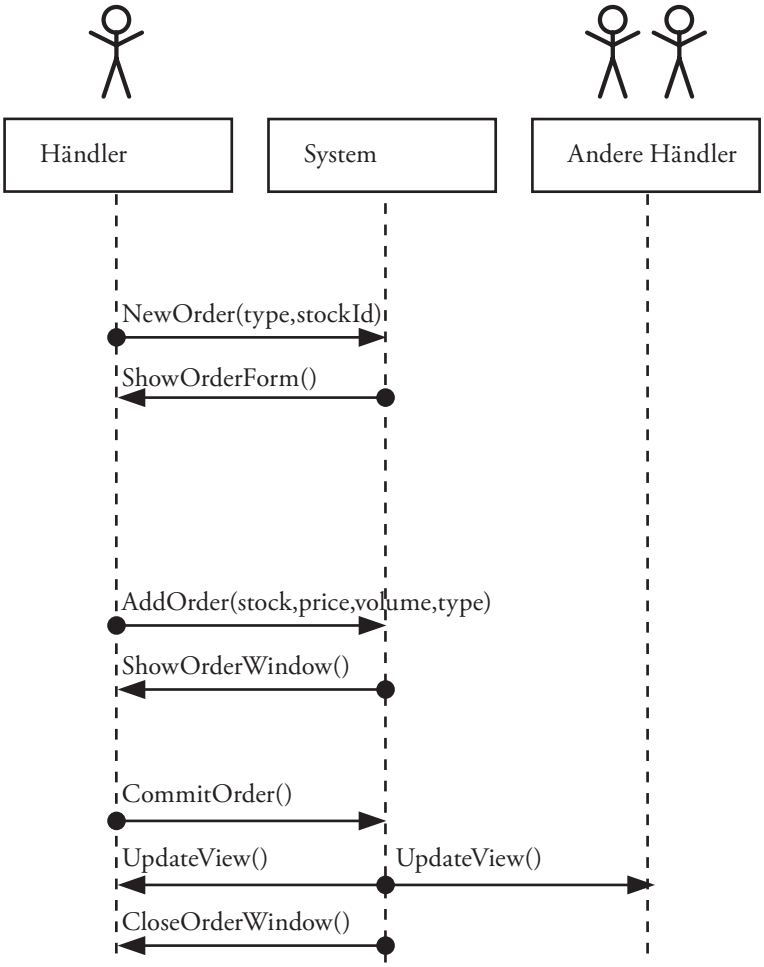
Attribute	Description
Identifier of use case and Name of use case	HA.001 Handle Aktien
Type of use case	Hauptanwendsungsfall
Short description / Goal of the use case	Der Händler setzt von der Handelsplattform einen Auftrag ab.
Actor(s)	Händler, System, andere Händler
Basic flow / steps (Use case drafts: define only the steps but no details or further explanations)	1. Der Händler wählt bei einer Aktie kaufen aus. (Bsp.: SVP) 2. Das System präsentiert eine Auftragsmaske mit drei Voreingestellten Optionen (Auftragart: Kaufen, Markt; Nationalratswahlen, Aktie: SVP) 3. Der Händler gibt Anzahl Aktien ein. 4. Der Händler gibt die Limite ein. 5. Der Händler bestätigt seine Eingabe. 6. Das System präsentiert erneut alle Auftragsdaten. (Bsp.: Auftragsart: Kaufen, Markt: Nationalratswahlen 2007, Aktien: SVP, Anzahl, Limite, Gesamtsumme Auftrag) 7. Der Händler bestätigt diesen Auftrag 8. System verarbeitet den Auftrag und trägt die Daten für andere Händler auf der Handelsplattform nach. 9. Die Eingabemaske verschwindet.

Alternative flow (Use case drafts: define only the alternative flows but not the actions taken)	A1 Der Händler wählt verkaufen aus. (Hauptanwendungsfall Schritt 1.) 1. Der Händler wählt bei einer Aktie verkaufen aus. (Bsp.: SVP) 2. Das System präsentiert eine Auftragsmaske mit drei Voreingestellten Optionen. (Auftragart: Verkaufen, Markt; Nationalratswahlen, Aktie: SVP) 3. Analog Hauptanwendungsfall Schritte 3.-5. 4. Das System präsentiert erneut alle Auftragsdaten. (Bsp.: Auftragsart: Verkaufen, Markt: Nationalratswahlen 2007, Aktien: SVP, Anzahl, Limite, Gesamtsumme Auftrag) 5. Zurück zum Hauptanwendungsfall Schritt 7. A2 Die Limite bzw. die Anzahl Aktien sind ungültig.(Hauptanwendungsfall Schritt 6) 1. Das System meldet dem Händler den Fehler und fordert ihn auf, die Eingabe zu korrigieren. 2. Zurück zum Hauptanwendungsfall Schritt 3. A3 Der Händler bestätigt den Auftrag nicht (Hauptanwendungsfall Schritte 6 & 7) 1. Zurück zum Hauptanwendungsfall Schritt 2. A4 Der Händler bricht die Auftragserfassung ab (Hauptanwendungsfall Schritte 3. –5.) 1. Zurück zum Hauptanwendungsfall Schritt 9. A5 Der Händler ändert die voreingestellten Optionen (Hauptanwendungsfall Schritte 3. – 5.) 1. Händler ändert Auftragsart und/oder Markt mit Aktie. 2. Zurück zum Hauptanwendungsfall Schritt 3.
Pre-conditions	Händler befindet sich auf der Handelsplattform
Post-conditions	- Der Auftrag führt zu einem kompletten Auftrag. - Der Händler hat Aktien im Depot und weniger Geld zur Verfügung - Der Auftrag wurde nicht ausgeführt, sondern ist im Börsenbuch hängig. - Der Auftrag wurde partiell ausgeführt, ein partieller Auftrag ist im Börsenbuch hängig.
Extension points	Hauptanwendungsfall Schritt 9.
Supplementary functional requirements	
Supplementary, non-functional requirements	

SSD

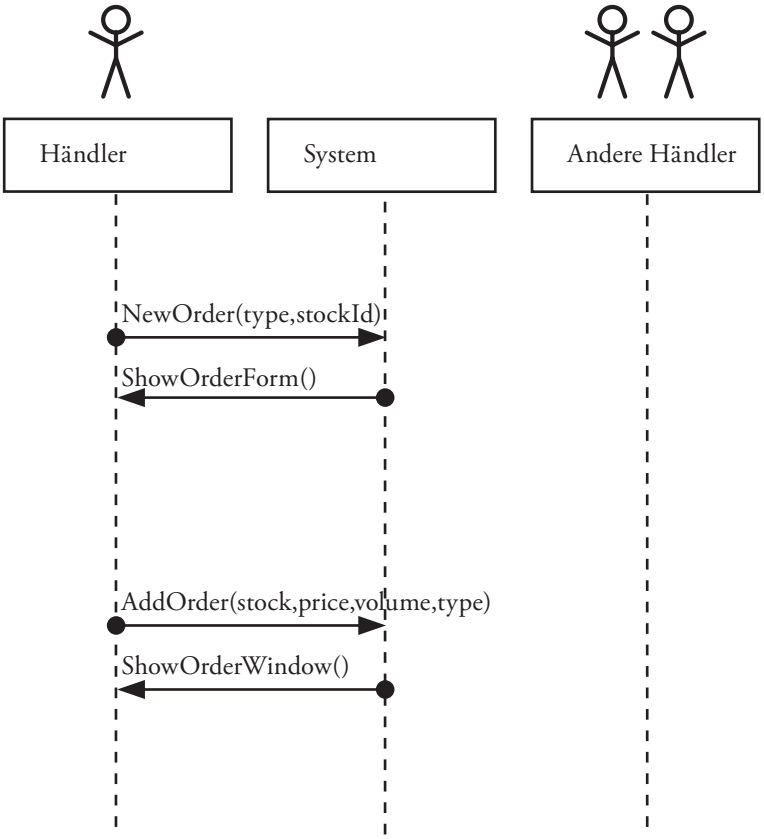
HA.001

1. Der Händler wählt bei einer Aktie kaufen aus. (Bsp.: SVP)
2. Das System präsentiert eine Auftragsmaske mit drei Voreingestellten Optionen (Auftragart: Kaufen, Markt; Nationalratswahlen, Aktie: SVP)
3. Der Händler gibt Anzahl Aktien ein.
4. Der Händler gibt die Limite ein.
5. Der Händler bestätigt seine Eingabe.
6. Das System präsentiert erneut alle Auftragsdaten. (Bsp.: Auftragsart: Kaufen, Markt: Nationalratswahlen 2007, Aktien: SVP, Anzahl, Limite, Gesamtsumme Auftrag)
7. Der Händler bestätigt diesenAuftrag
8. System verarbeitet den Auftragund trägt die Daten für andere Händler auf der Handelsplattform nach.
9. Die Eingabemaske verschwindet.



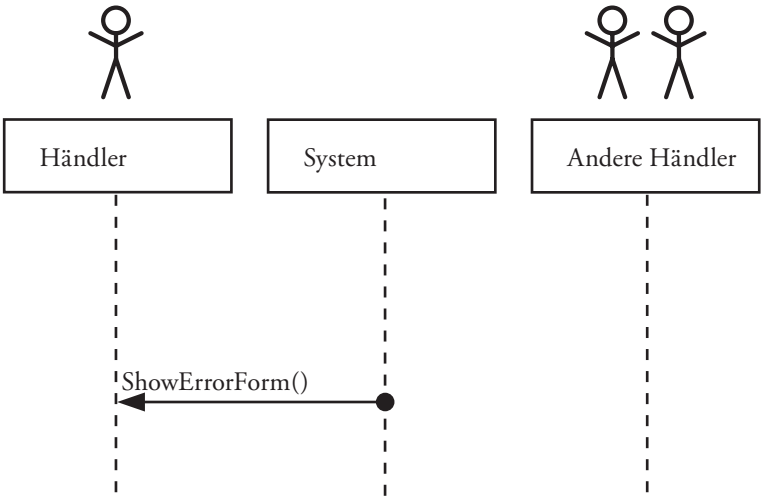
Alternative A1

1. Der Händler wählt bei einer Aktie verkaufen aus. (Bsp.: SVP)
2. Das System präsentiert eine Auftragsmaske mit drei Voreingestellten Optionen. (Auftragart: Verkaufen, Markt; Nationalratswahlen, Aktie: SVP)
3. Analog Hauptanwendungsfall Schritte 3.-5.
4. Das System präsentiert erneut alle Auftragsdaten. (Bsp.: Auftragsart: Verkaufen, Markt: Nationalratswahlen 2007, Aktien: SVP, Anzahl, Limite, Gesamtsumme Auftrag)
5. Zurück zum Hauptanwendungsfall Schritt 7.



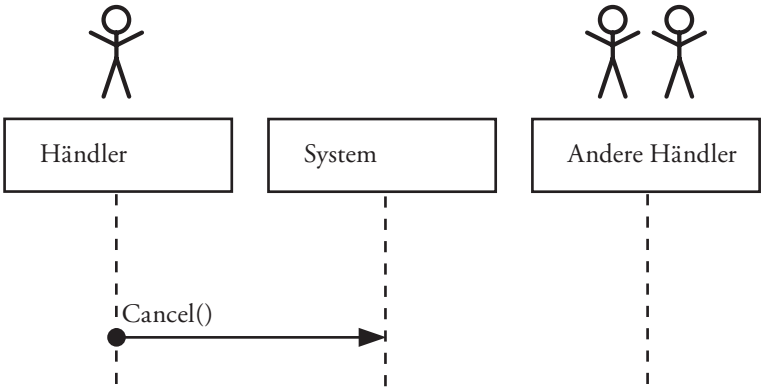
Alternative A2

1. Das System meldet dem Händler den Fehler und fordert ihn auf, die Eingabe zu korrigieren.
2. Zurück zum Hauptanwendungsfall Schritt 3.



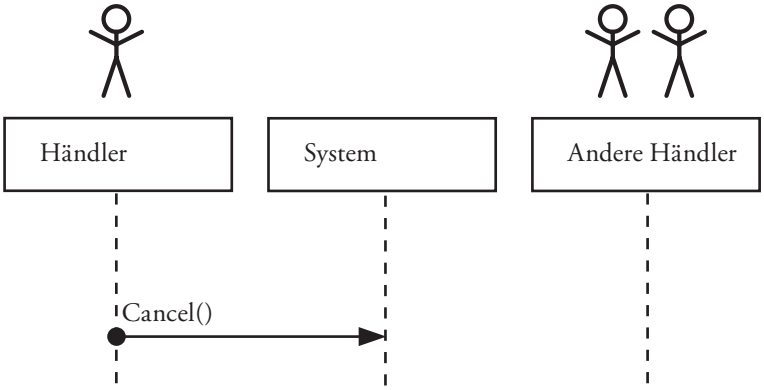
Alternative A3

1. Zurück zum Hauptanwendungsfall Schritt 2.



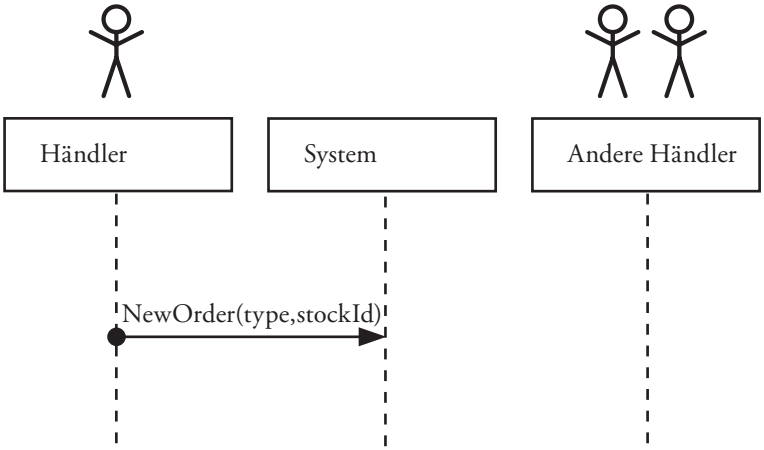
Alternative A4

1. Zurück zum Hauptanwendungsfall Schritt 9.



Alternative A5

1. Händler ändert Auftragsart und/oder Markt mit Aktie.
2. Zurück zum Hauptanwendungsfall Schritt 3.



Operation Contracts (HA.001)

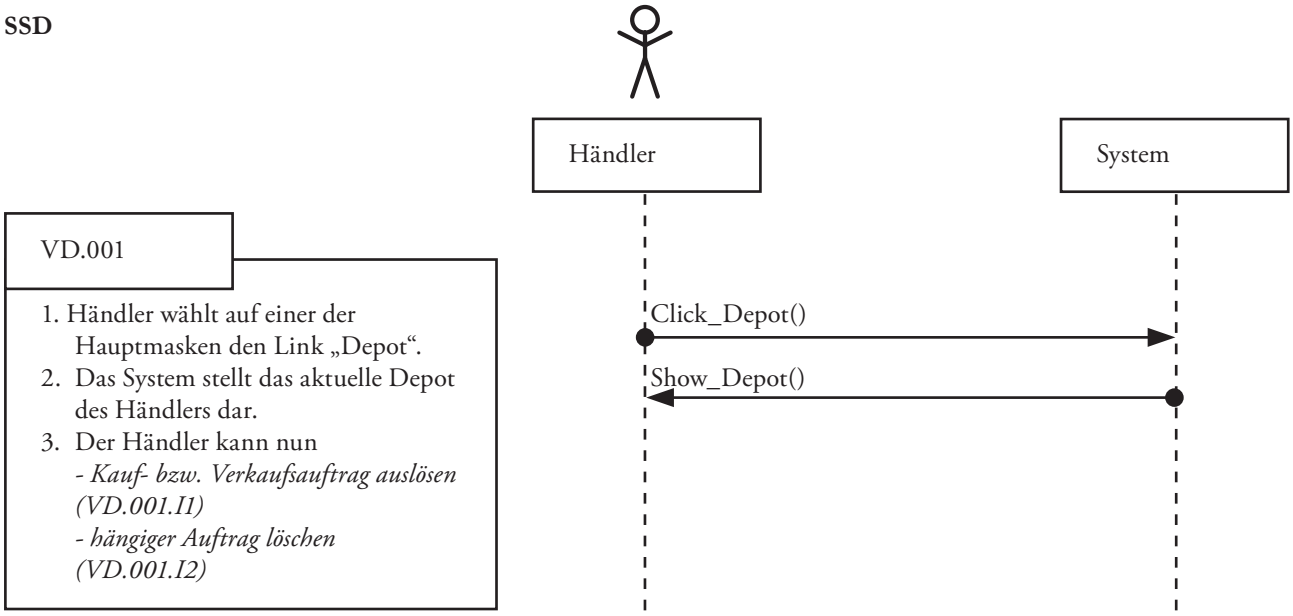
Operation:	NewOrder(type,stockID)
Pre conditions:	Der Händler ist eingeloggt.
Post conditions:	Das System überprüft ob der Händler noch eingeloggt ist.
Operation:	ShowOrderForm()
Pre conditions:	Der Händler ist eingeloggt und er hat auf “kaufen“ bzw. “verkaufen“ gedrückt.
Post conditions:	System öffnet ein Fenster mit einer Auftragseingabemaske.
Operation:	TransmitData(type, stockID, quantity, limit)
Pre conditions:	Händler hat in die Eingabemaske alle erforderlichen Felder ausgefüllt.
Post conditions:	System hat alle Eingaben entgegengenommen. Eingabemaske bleibt offen.
Operation:	ShowOrderWindow()
Pre conditions:	Der Händler hat schon einen vollständigen Auftrag aufgegeben.
Post conditions:	System zeigt den Auftrag zur Bestätigung nochmals an.
Operation:	CommitOrder()
Pre conditions:	Auftrag wurde schon vollständig eingegeben. Auftragsbestätigung ist sichtbar.
Post conditions:	Der Händler hat den Auftrag bestätigt. Das System verarbeitet den Auftrag. System schliesst Bestätigungsfenster.
Operation:	UpdateView()
Pre conditions:	Auftrag wurde vom Händler bestätigt.
Post conditions:	Die Handelsübersichtseite wurde neu geladen mit den neuen Werten.
Operation:	CloseOrderWindow()
Pre conditions:	Auftrag wurde bestätigt, Bestätigungsfenster wurde geschlossen und Handelsübersichtseite ist neu geladen.
Post conditions:	Auftragseingabemaske wurde geschlossen.

5.5.3 Verwalte Depot (VD.001)

Use case

Attribute	Description
Identifier of use case and Name of use case	VD.001 Verwalte Depot
Type of use case	Hauptanwendungsfall
Short description / Goal of the use case	Der Händler öffnet sein Depot. Er kann einerseits über die Übersicht handeln wie auch hängige Aufträge löschen.
Actor(s)	Händler, System
Basic flow / steps (Use case drafts: define only the steps but no details or further explanations)	1. Händler wählt auf einer der Hauptmasken den Link „Depot“. 2. Das System stellt das aktuelle Depot des Händlers dar. 3. Der Händler kann nun *- Kauf- bzw. Verkaufsauftrag auslösen (VD.001.I1)* *- hängiger Auftrag löschen (VD.001.I2)*
Alternative flow (Use case drafts: define only the alternative flows but not the actions taken)	
Pre-conditions	Händler befindet sich auf einer „Hauptmaske“
Post-conditions	VD.001.I1, VD.001.I2
Extension points	
Supplementary functional requirements	
Supplementary, non-functional requirements	

SSD


Händler
System
VD.001
1. Händler wählt auf einer der Hauptmasken den Link „Depot“.
2. Das System stellt das aktuelle Depot des Händlers dar.
3. Der Händler kann nun
- Kauf- bzw. Verkaufsauftrag auslösen (VD.001.I1)
- hängiger Auftrag löschen (VD.001.I2)
Click_Depot()
Show_Depot()

5.5.4 Kauf- bzw. Verkauf auslösen (VD.001.I1)

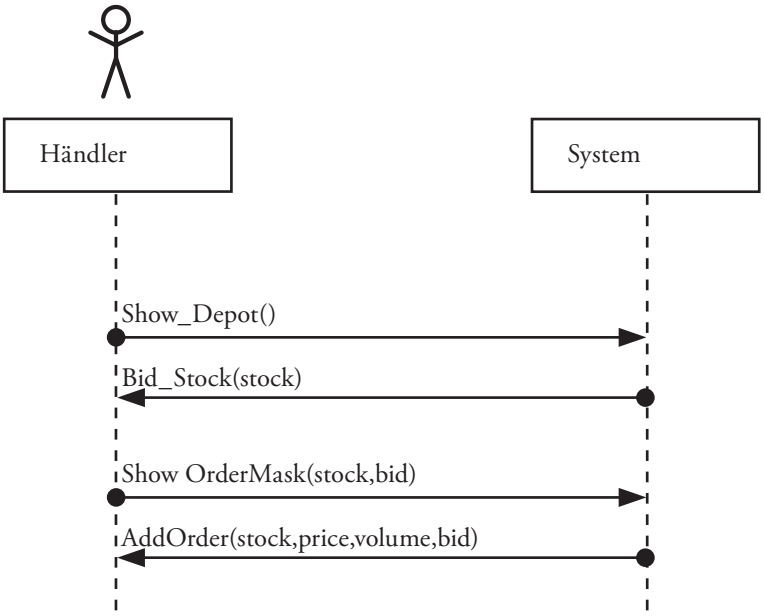
Use case

Attribute	Description
Identifier of use case and Name of use case	VD.001.I1 VD Handeln
Type of use case	(1) inclusion use case
Short description / Goal of the use case	Der Händler handelt mit den Aktien in seinem Depot.
Actor(s)	Händler, System
Basic flow / steps (Use case drafts: define only the steps but no details or further explanations)	1. Das System listet alle Aktien, wie auch das restlich vorhandene Geld im Besitz des Händlers auf. 2. Der Händler wählt "verkaufen" bei einer vorhandenen Aktie im Depot. 3. Das System zeigt eine angepasste Auftragsmaske. 4. Der Händler gibt Volumen und Preis an und bestätigt die Eingabe.
Alternative flow (Use case drafts: define only the alternative flows but not the actions taken)	A1 Der Händler wählt "kaufen" in der Depotübersicht. (Hauptanwendungsfall Schritt 1) 1. Der Händler wählt in der Depotübersicht "kaufen". 2. Das System zeigt angepasst Auftragsmaske an. 3. Der Händler wählt Aktie zum Kaufen aus, gibt Volumen und Preis an und bestätigt Eingabe. A2 Der Händler gibt einen ungültigen Auftrag ein. (Hauptanwendungsfall Schritt 1 und Alternative A1 Schritt1) 1. Das System zeigt angepasste Auftragsmaske. 2. Der Händler gibt einen ungültigen Auftrag ein. 3. Das System wirft Fehlermeldung. 4. Zurück zum Hauptanwendungsfall Schritt 4 und Alternative A1 Schritt 3.
Pre-conditions	Der Händler befindet sich auf einer „Hauptmaske"
Post-conditions	Kauf- bzw. Verkaufsauftrag wurde vom Auftragsbuch entgegengenommen
Extension points	
Supplementary functional requirements	
Supplementary, non-functional requirements	

SSD

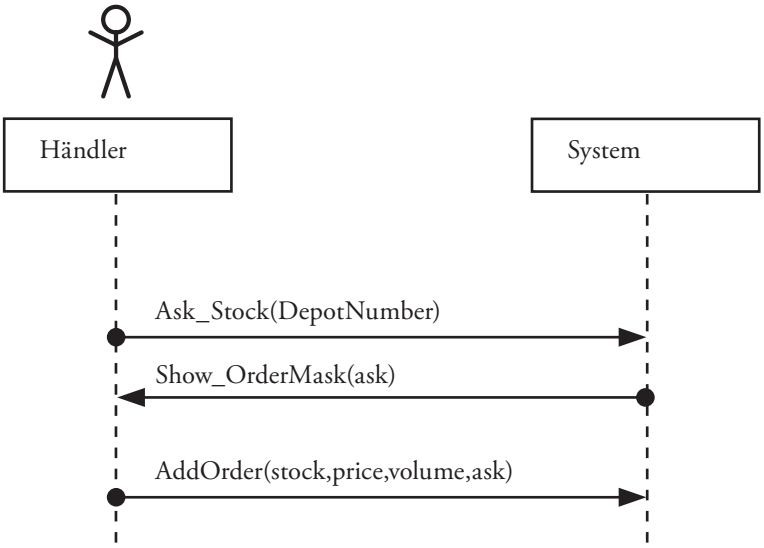
VD.001.I1

1. Das System listet alle Aktien, wie auch das restlich vorhandene Geld im Besitz des Händlers auf.
2. Der Händler wählt "verkaufen" bei einer vorhandenen Aktie im Depot.
3. Das System zeigt eine angepasste Auftragsmaske.
4. Der Händler gibt Volumen und Preis an und bestätigt die Eingabe.



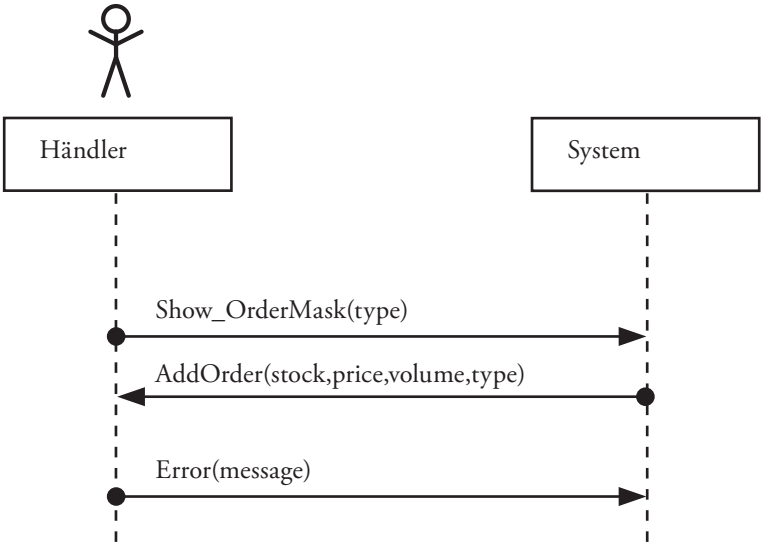
Alternative A1

1. Der Händler wählt in der Depotübersicht "kaufen".
2. Das System zeigt angepasst Auftragsmaske an.
3. Der Händler wählt zum Kaufen aus, gibt Volumen und Preis und bestätigt die Eingabe.



Alternative A2

1. Das System zeigt angepasste Auftragsmaske.
2. Der Händler gibt einen ungültigen Auftrag ein.
3. Das System wirft Fehlermeldung.
4. Zurück zum Hauptanwendungsfall Schritt 4 und Alternative A1 Schritt 3.



5.5.5 Lösche Auftrag (VD.001.I2)

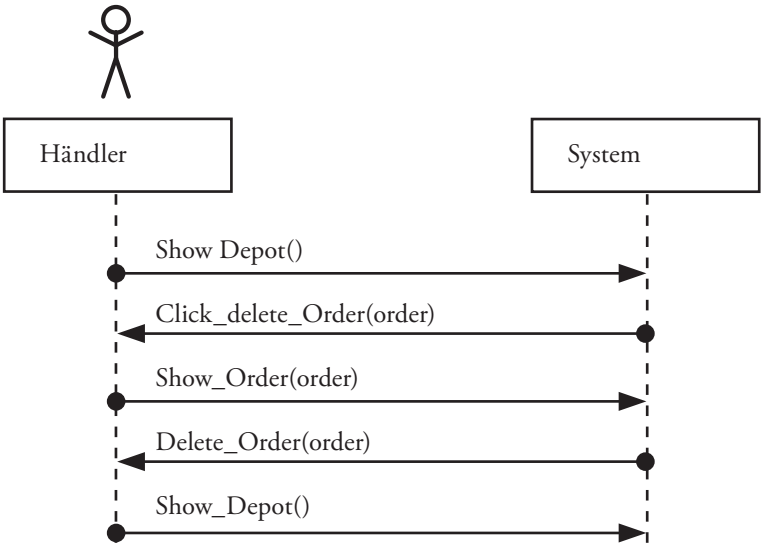
Use case

Attribute	Description
Identifier of use case and Name of use case	VD.001.I2 VD Löschen
Type of use case	(1) inclusion use case
Short description / Goal of the use case	Der Händler löscht ein Auftrag
Actor(s)	Händler, System
Basic flow / steps (Use case drafts: define only the steps but no details or further explanations)	1. Das System zeigt hängige Aufträge an. 2. Der Händler wählt „löschen“. 3. Das System zeigt diesen Auftrag an. 4. Der Händler bestätigt Löschung. 5. Das System zeigt die neue Depotübersicht.
Alternative flow (Use case drafts: define only the alternative flows but not the actions taken)	A1 Der Auftrag ist zwischenzeitlich nicht mehr vorhanden (Hauptanwendungsfall Schritt 5) 1. Das System meldet, dass der Auftrag nicht mehr vorhanden ist 2. Zurück zum Hauptanwendungsfall Schritt 1.
Pre-conditions	Der Händler befindet sich auf einer „Hauptmaske“
Post-conditions	Auftrag ist nicht mehr vorhanden

SSD

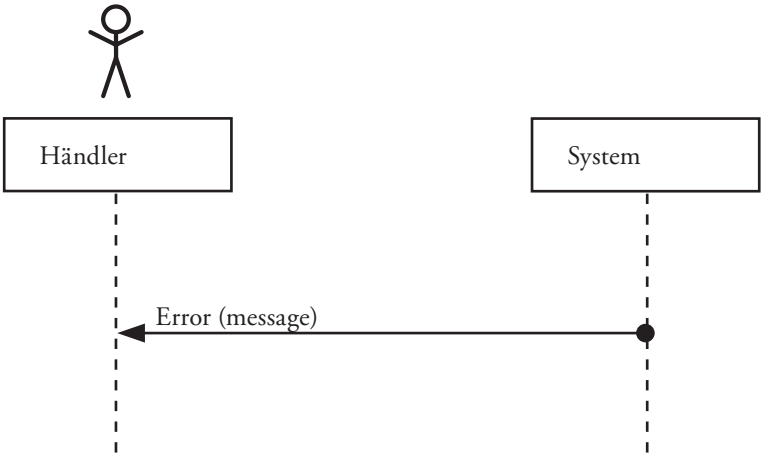
VD.001.I2

1. Das System zeigt hängige Aufträge an
2. Der Händler wählt „löschen“
3. Das System zeigt diesen Auftrag an
4. Der Händler bestätigt Löschung
5. Das System zeigt die neue Depotübersicht.



Alternative A1

1. Das System meldet, dass der Auftrag nicht mehr vorhanden ist
2. Zurück zum Hauptanwendungsfall Schritt 1.



5.5.6 Anmelden (AN.001)

Use Case

Attribute	Description
Identifier of use case and Name of use case	AN.001 Anmelden
Type of use case	Hauptanwendsungsfall
Short description / Goal of the use case	Ein neuer Benutzer meldet sich an.
Actor(s)	Neuer Benutzer, System
Basic flow / steps (Use case drafts: define only the steps but no details or further explanations)	1. Der neue Benutzer befindet sich auf der Anmeldemaske und gibt seine Daten ein. 2. System validiert die Eingaben und gibt ein positives Feedback. 3. System sendet dem neuen Benutzer eine E-Mail mit einem Freischaltcode. 4. Der neue Benutzer schaltet über die erhaltene E-Mail seinen Benutzername Frei. 5. System kreiert neuen Händler mit Startguthaben.
Alternative flow (Use case drafts: define only the alternative flows but not the actions taken)	A1 Eingegebene Daten in der Anmeldemaske sind nicht zulässig. (Hauptanwendungsfall Schritt 2.) 1. Das System validiert die Eingaben und gibt ein negatives Feedback. 2. Zurück zum Hauptanwendungsfall Schritt 1.
Pre-conditions	Benutzer noch nicht angemeldet.
Post-conditions	Neuer Benutzer angemeldet. Username ist frei geschaltet.

5.5.7 Login (LG.001)

Use Case

Attribute	Description
Identifier of use case and Name of use case	LG.001 Login
Type of use case	Hauptanwendungsfall
Actor(s)	Händler, System
Basic flow / steps (Use case drafts: define only the steps but no details or further explanations)	Sobald der Benutzer auf einen geschützten Bereich auf dem System gelangen will, wird er aufgefordert sich zu authentisieren. Die Authentisierung geschieht über Username und Password. Das System authentisiert den Benutzer über die Angaben in der Datenbank.
Alternative flow (Use case drafts: define only the alternative flows but not the actions taken)	Händler hat sich noch nicht eingeloggt
Pre-conditions	Der Händler ist eingeloggt. Er kann nun auf geschützte Bereiche zugreifen.
Post-conditions	

5.6 Beans Konzept

Wir haben uns nach der dritten Iteration entschieden, die DataSets für den Datenbank-Zugriff aussen vor zu lassen, da wir beim Iterieren über DataViews ab und zu Fehler bekommen haben, was wahrscheinlich ein Fehler in der DataView-Klassen ist. Der Fehler trat nämlich nicht immer auf und er nahm unterschiedliche Ausmasse an. Ab und zu lieferte uns das Iterieren mit der foreach-Schlaufe eine Zeile, mal keine und häufig die richtige Anzahl Zeilen. Wir konnten durch Tests feststellen, dass wir tatsächlich nichts am DataSet geändert haben und durch mehrfaches Wiederholen des Iterieren durch die DataViews unterschiedliche Ergebnisse erhielten.

Da wir unser Vertrauen in DataSets nun verloren haben und sie für unsere Zwecke zu umständlich schienen, haben wir ein Konzept der Beans entworfen (Name als Anlehnung zu J2EE-Entity-Beans).

Ein Bean ist eine eingepackte Datenbankzeile (Tupel). Ein Bean implementiert folgende Schnittstelle: IBean. Es muss also wissen, zu welcher Tabelle es gehört und es muss in der Lage sein, seine Felder als ein Object-Array abzuspeichern und das Mapping mithilfe von einer Methode ColumnNames() zur Verfügung zu stellen. Ebenso hat ein Bean immer eine ID. Die restlichen Felder werden mit Properties zur Verfügung gestellt, so dass die Businesslogik in der Lage ist, das Bean zu verändern.

Das Bean kann von der Business-Logik mit einem normalen Konstruktor erstellt werden. Für den TransactionHandler, der Beans von der Datenbank generieren muss, wird eine Factory zur Verfügung gestellt, die die IBeanFactory-Schnittstelle zur Verfügung stellt. Dadurch kann ein Tupel (als Object-Array) in ein bestimmtes Bean umgewandelt werden. Der TransactionHandler stellt Methoden zur Verfügung, um einzelne Beans oder als Collection zu laden und zu speichern.

TransactionHandler:

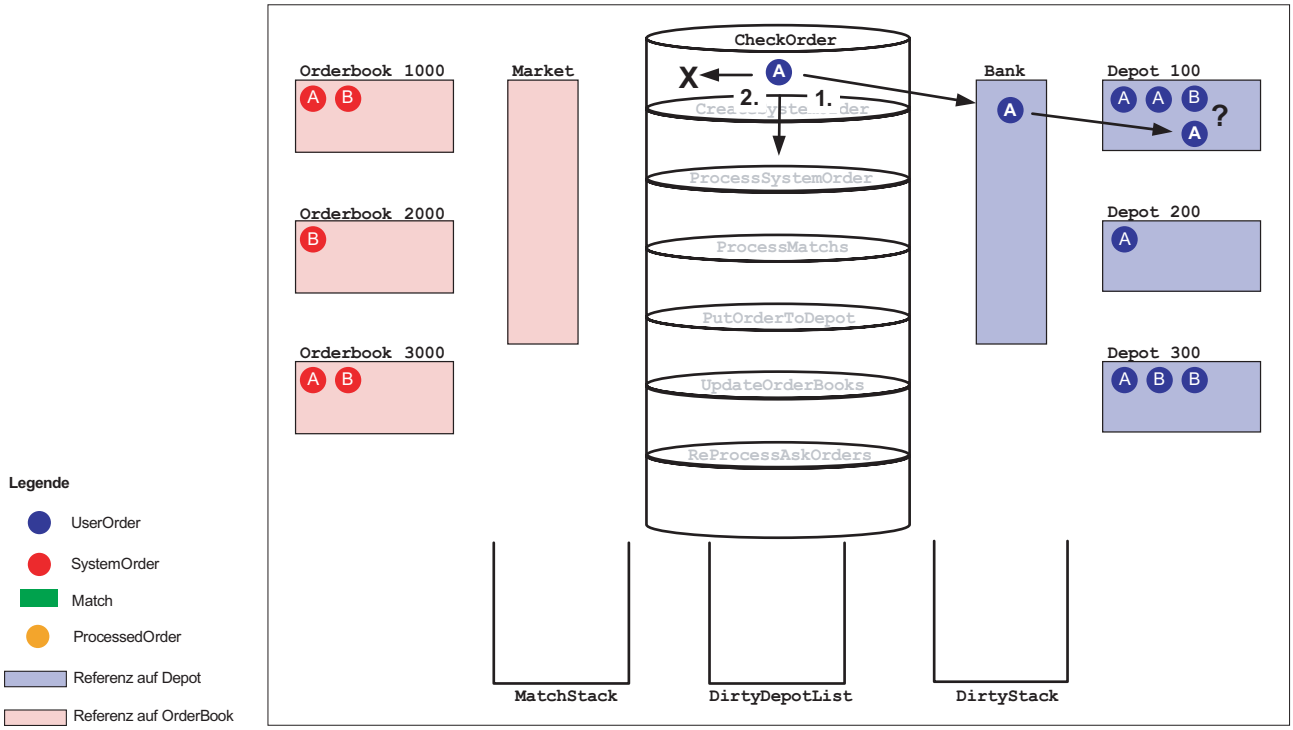
```
public ICollection GetBeanCollection(string identifierName,
object identifier, IBeanFactory factory);

public ICollection GetBeanCollection(string[] identifierNames,
object[] identifiers, IBeanFactory factory);

public object GetBean(IBeanFactory factory);

public object GetBean(IBeanFactory factory,string sqlStatement);

public object GetBean(string identifierName, object identifier,
IBeanFactory factory);

public object GetBean(string[] identifierNames, object[]
identifiers, IBeanFactory factory);
```

Wir können nun die Datenbank-Tupel flexibel handhaben. Für unsere Lösung haben wir zwei Sorten von Beans unterschieden und in zwei Namespaces getan: MarketBeans und ViewBeans. MarketBeans sind dabei Beans, die primär von der Businesslogik gebraucht werden, sie werden aber auch von der View verwendet. ViewBeans dagegen werden ausschliesslich von der View verwendet.

6 Design

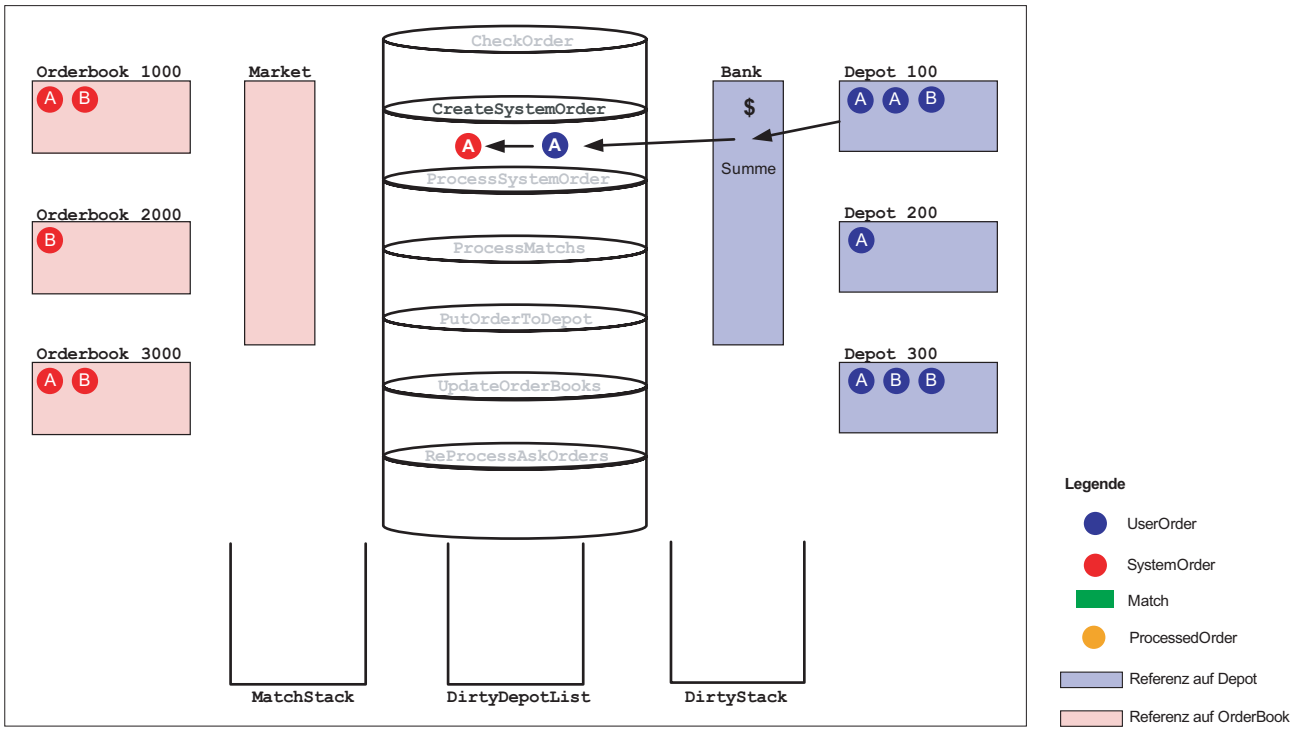
6.1 Typische Transaktion in der Klasse TransactionController

6.1.1 CheckOrder



Bei der Stufe CheckOrder wird zuerst einmal überprüft, ob sich im Depot ein Auftrag befindet, der der Regel widerspricht, dass der Benutzer keine Aufträge absetzen darf, die selber zu einem Handel führen. Bei einem neuen Kaufauftrag darf also kein Verkaufsauftrag bereits im Depot sein, dessen Limite tiefer oder gleich gross ist. Bei einem neuen Verkaufsauftrag darf kein Kaufauftrag im Depot sein, dessen Limite grösser oder gleich dem neuen Auftrag ist. Wenn die Regel verletzt ist, wird das System dem neuen Auftrag fortwerfen. Ansonsten wird er den Auftrag an die nächste Stufe weiterreichen.

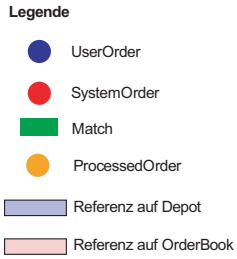
6.1.2 CreateSystemOrder

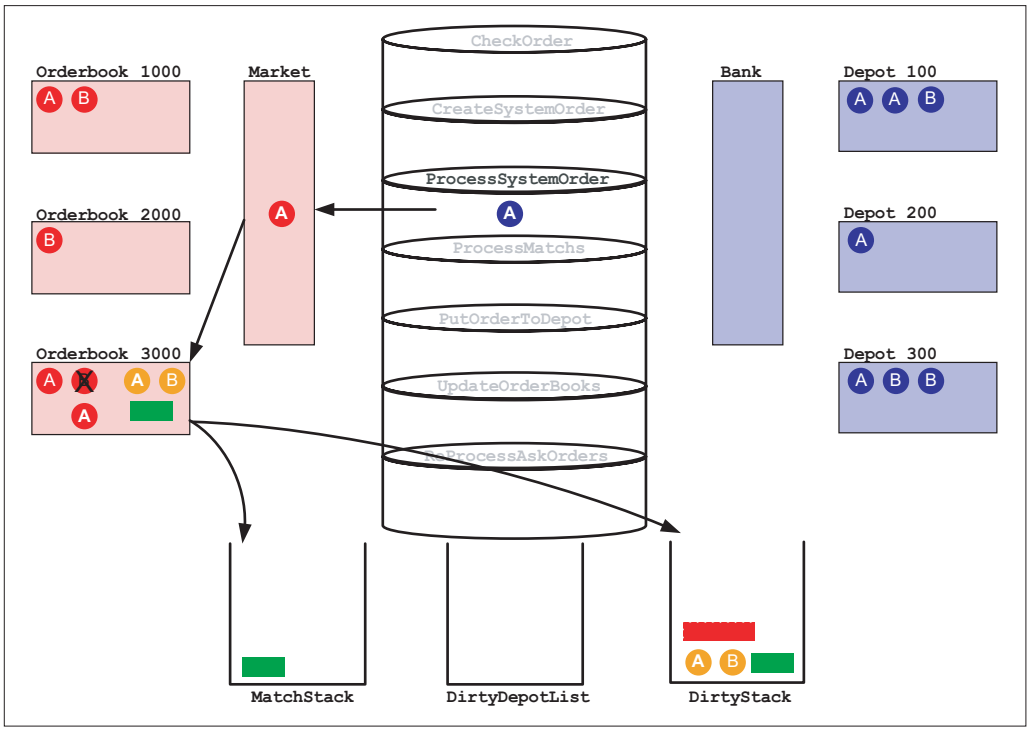


Bei der Stufe CreateSystemOrder wird in einem ersten Schritt einmal das UserOrder-Bean als SystemOrder-Bean geklont (d.h. alle Felder werden 1 zu 1 übernommen, ausser, dass das Bean nun einem Tupel der Tabelle „SystemOrder“ entspricht.).

Als zweiten Schritt wird nun je nach Auftrag die maximal zur verfügbaren Aktien (bei „bid“) oder das maximal verfügbare Geld (bei „ask“) über das Depot ermittelt und das SystemOrder-Bean so angepasst, dass es zu 100% ausführbar ist. Hat der Benutzer dagegen keine Aktien oder kein Geld, so werden die folgenden Stufen ProcessSystemOrder, ProcessMatchs einfach übersprungen.

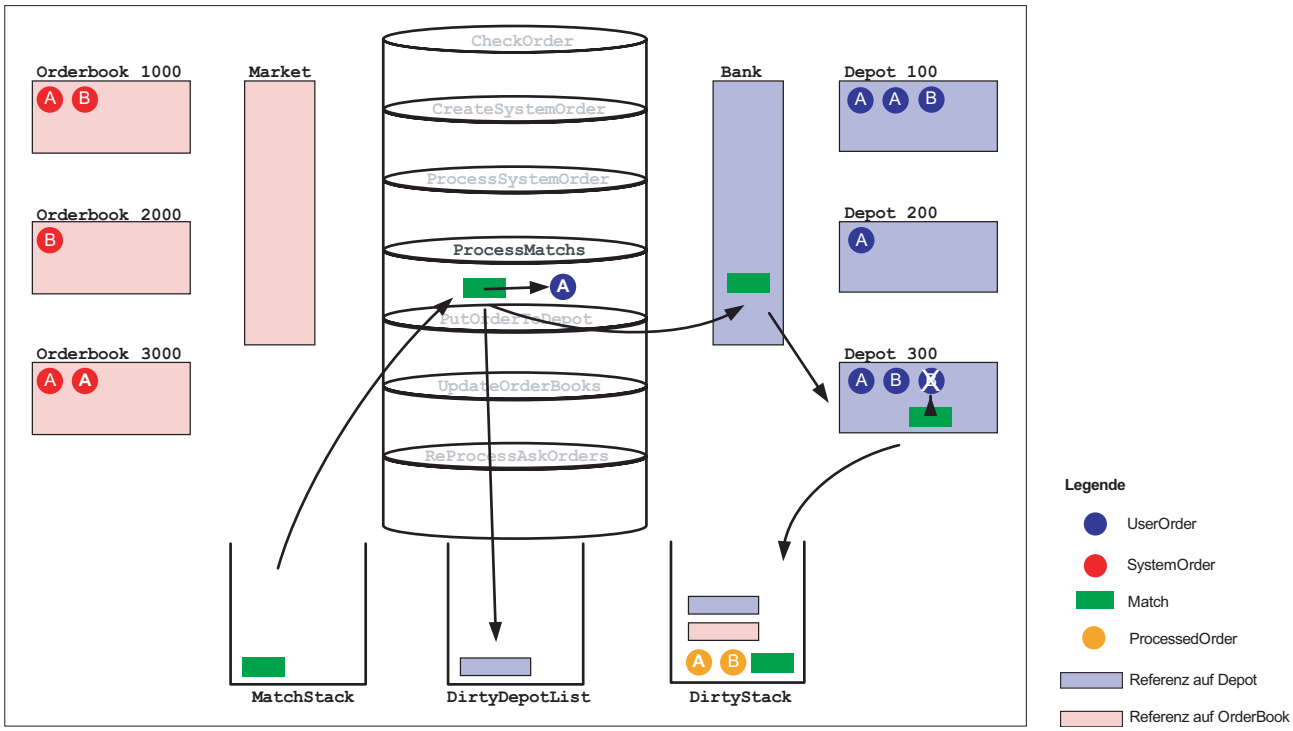
6.1.3 ProcessSystemOrder





Bei der Stufe ProcessSystemOrder wird das Bean über den Market in das entsprechende OrderBook geleitet, wo die Aufträge für die Aktien verwaltet werden. Hier kann nun das Bean zu einem oder mehreren Matches führen. D.h. es werden SystemOrder im Volumen verändert oder ganz eliminiert. Bei jedem Match wird ein Match-Bean erzeugt, welches dem MatchStack zugeführt wird. Ebenso werden von allen Order-Beans, die an einem Match beteiligt wurden, ProcessOrder-Beans kreiirt. Sowohl die ProcessOrder-Beans, wie auch die Match-Beans werden auf den DirtyStack gelegt, damit die Matches auf der Datenbank protokolliert werden. Am Schluss legt auch das OrderBook noch eine Referenz auf sich auf dem DirtyStack ab, weil die SystemOrder-Beans verändert oder gelöscht wurden und dies nun persistent gemacht werden muss.

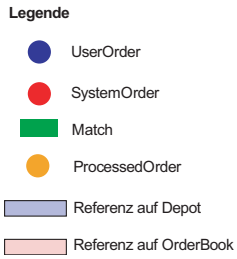
6.1.4 ProcessMatch

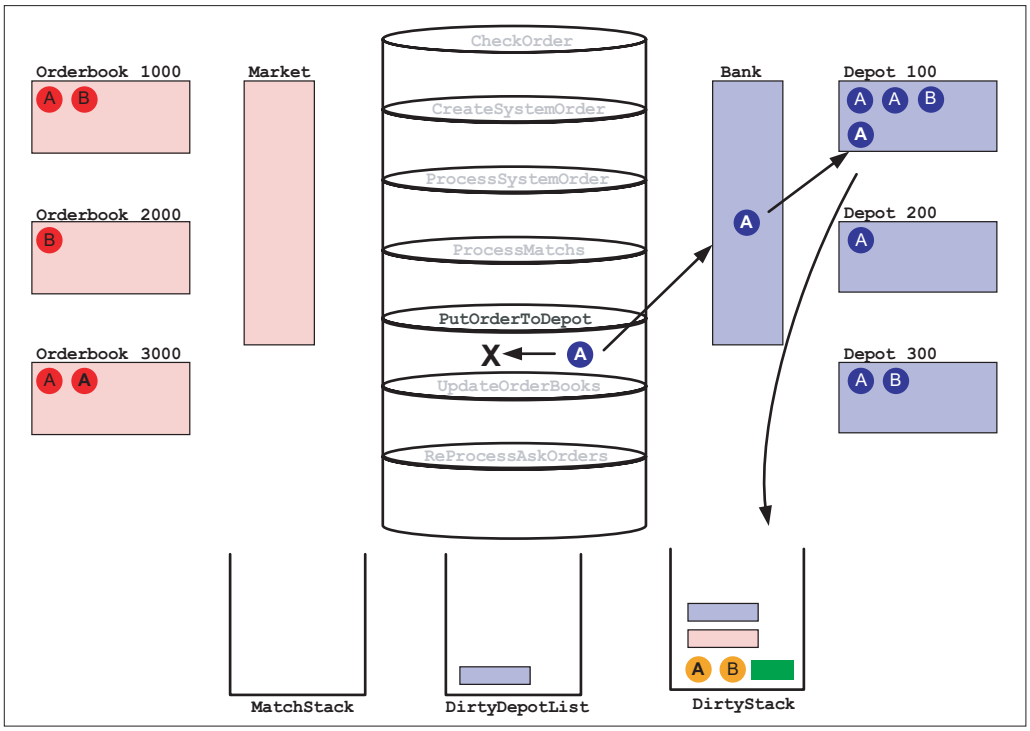


Da die UserOrder-Beans nun nicht mehr konsistent gegenüber den SystemOrder-Beans sind, müssen nun die UserOrder-Beans angepasst werden. Dazu holt sich der TransactionController die Match-Beans vom MatchStack und passt den neuen UserOrder-Bean an. Ebenso merkt er sich die ID des Depots, von wo der gematchte Order stammt in der DirtyDepotList. Er gibt das Match-Bean schlussendlich an die Bank weiter, wo es in das Depot des gematchten UserOrders geleitet wird. Das Depot passt nun ebenfalls den UserOrder an und legt eine Referenz auf sich selber auf dem DirtyStack ab.

Dieser Vorgang wird nun für jeden Match wiederholt, bis der MatchStack leer ist.

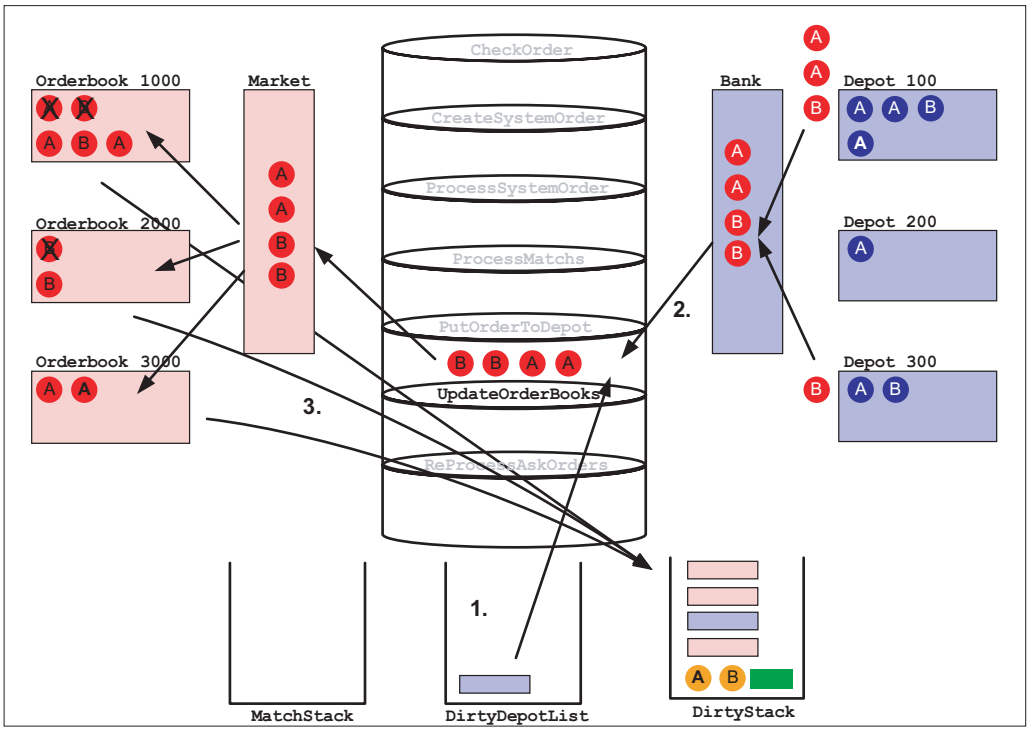
6.1.5 PutOrderToDepot





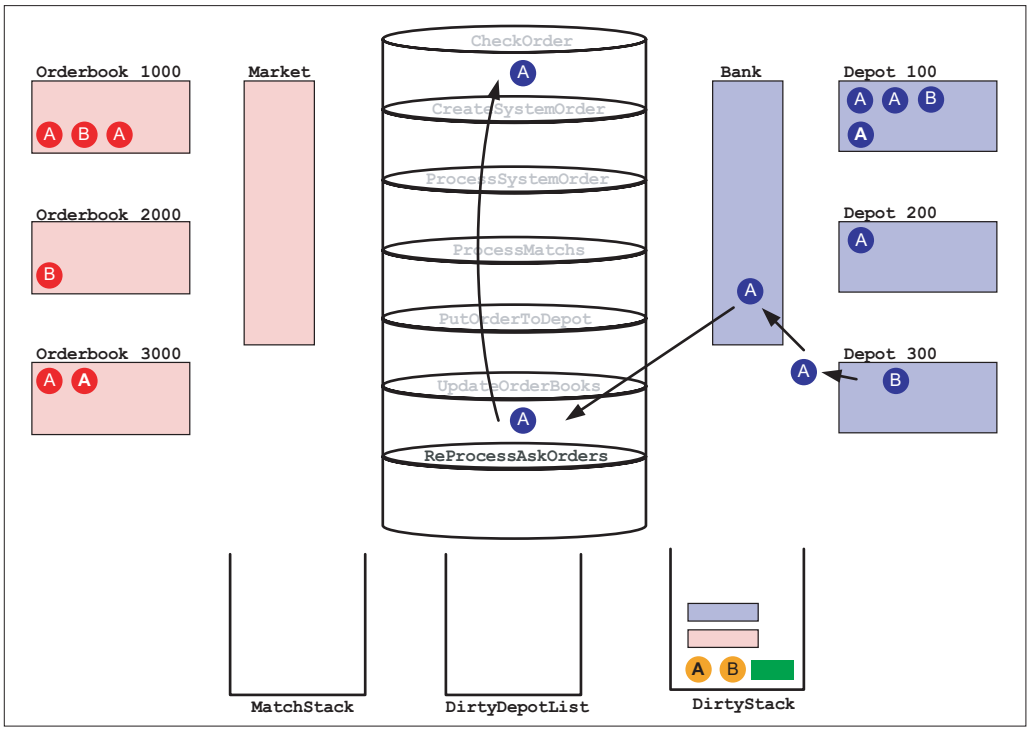
Bei der Stufe PutOrderToDepot wird der angepasste Order ins Depot ausgelagert. Ausser wenn genauso viel Volumen gehandelt wurde, wie es im Order vorgesehen war, dann wird der Order einfach weggeworfen.

6.1.6 UpdateOrderBook



In dieser Stufe werden nun die SystemOrder-Beans in den OrderBooks angepasst. Da der Benutzer nun mehr Aktien oder mehr Geld hat, muss überprüft werden, ob ein UserOrder nun in einen neuen SystemOrder umgewandelt werden muss. Es gibt SystemOrder, die können einfach angepasst werden, ohne dass es zu Matches führt. Dazu beauftragt der TransactionController die Depots, die an den vergangenen Matches beteiligt waren, aus den UserOrders aktuelle SystemOrders zu erstellen. Er leitet dann diese zum Market, wo sie den OrderBooks übergeben werden. Die OrderBooks ersetzen nun die SystemOrders und legen schliesslich eine Referenz auf dem DirtyStack ab.

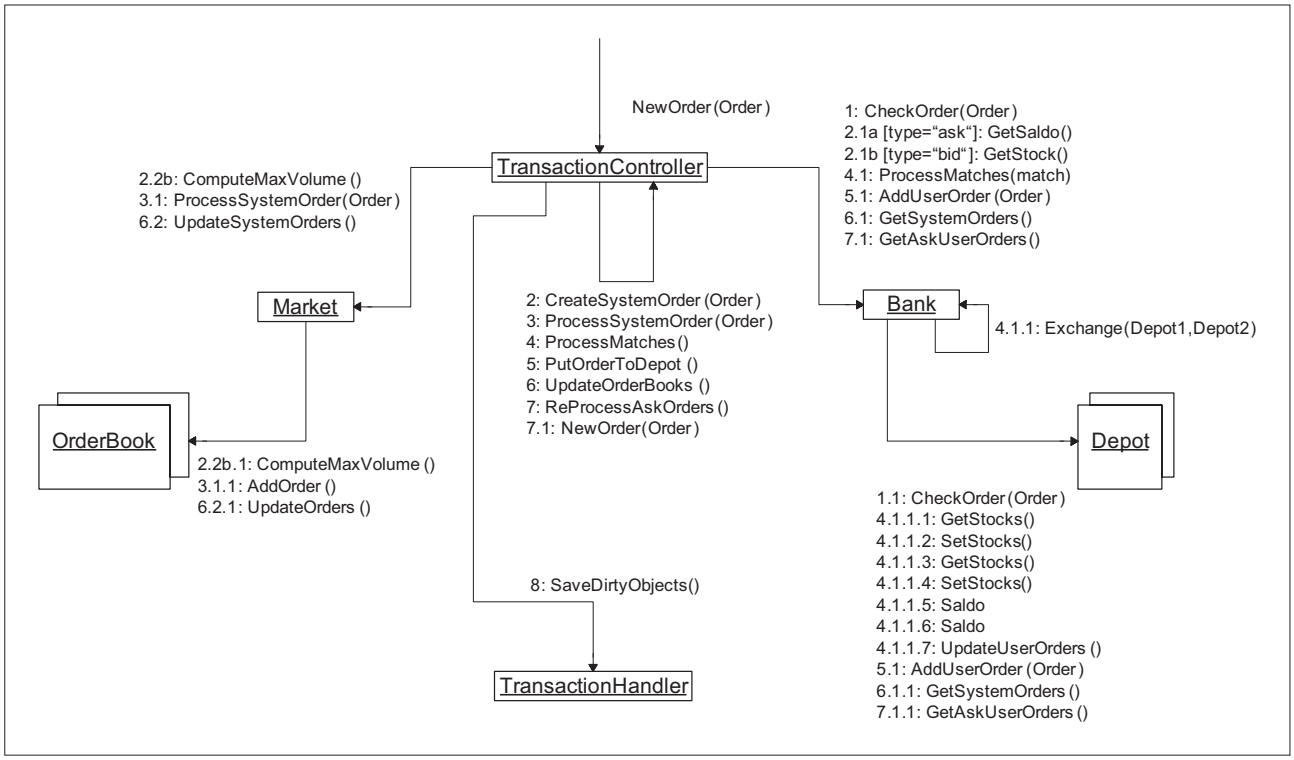
6.1.7 ReProcessAskOrder



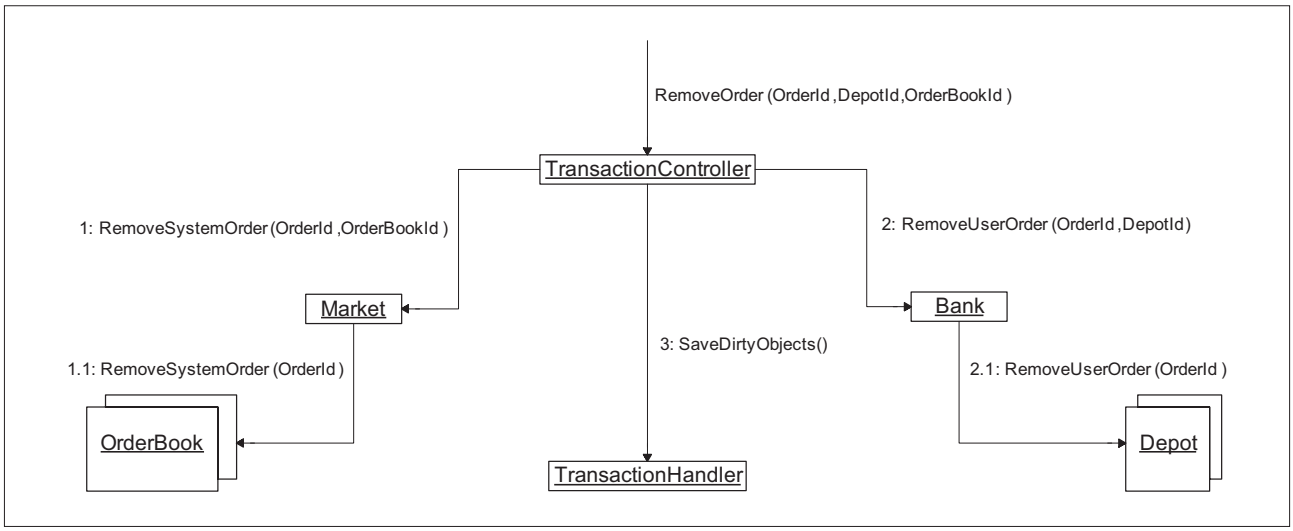
In dieser Stufe werden nun aus den Depots die UserOrder-Beans geholt, die zu neuen Matches führen können. Sie werden damit komplett aus dem Depot entfernt und neu in die Methode NewOrder hineingeworfen. Dadurch läuft das ganze rekursiv ab, bis es keine Asks mehr hat, die noch matchen könnten. Wenn keine Matches mehr in den MatchStack abgelegt werden, und sonst keine Einträge mehr in der DirtyDepotList stehen, dann führt das zum Abbruch der rekursiven Aufrufe.

6.2 Kollaborationsdiagramme

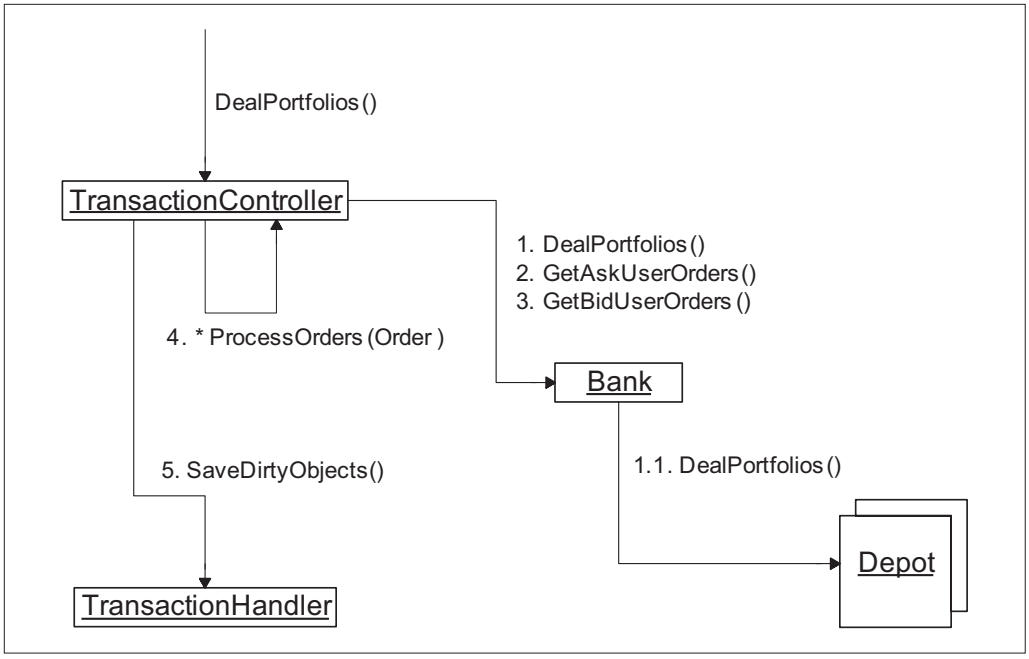
6.2.1 NewOrder(Order)



6.2.2 RemoveOrder(OrderId,DepotId,OrderbookId)

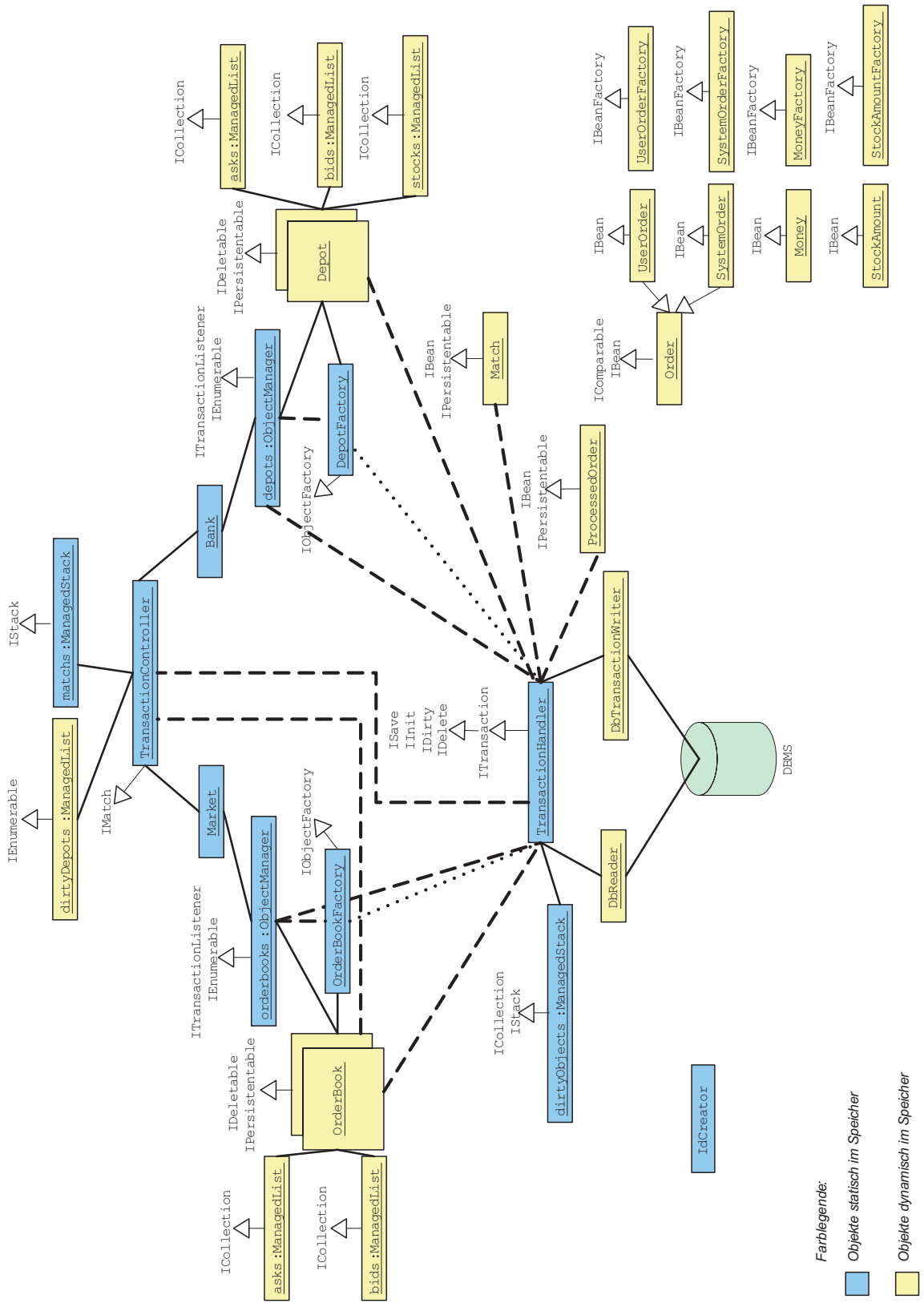


6.2.3 DealPortfolio()



6.3 Klassendiagramme

6.3.1 Kern der Business-Logik



6.4 Klassen des Kerns

6.4.1 TransactionController

Der `TransactionController` wird einmal instanziert und ist verantwortlich für den Ablauf einer Transaktion. (Er ist der aktive Teil einer Transaktion im Gegensatz zum `TransactionHandler`, welcher die passive Funktion übernimmt.) Eine Transaktion kann sein, dass jemand einen neuen Auftrag eingibt, Portfolios kauft oder verkauft oder einen Auftrag storniert. Der `TransactionController` beinhaltet zusätzlich aber noch einen Robot, der automatisch Portfolios handelt, um den Markt auf den gewünschten Durchschnittswert (100) zu korrigieren.

6.4.2 TransactionHandler

Der `TransactionHandler` ist der passive Teil einer Transaktion. Er ist verantwortlich, dass alle veränderten Objekte während einer Transaktion persistent gemacht werden und dass die Daten am Schluss konsistent auf der Datenbank stehen. Dazu stellt der `TransactionHandler` die `ITransaction`-Schnittstelle zur Verfügung. Über diese Schnittstelle können sich Objekte in die Dirty-List (als Stack implementiert) eintragen und auch über die Schnittstelle sich Initiieren, Speichern, Löschen oder auch sich bloss als `TransactionListener` eintragen. Der `TransactionHandler` bekommt vom `TransactionController` den Auftrag, dass er die Transaktion abschliessen soll, d.h. persistent und konsistent machen soll.

6.4.3 Market

`Market` hat eine Fassadenfunktion. `Market` verwaltet alle Auftragsbücher (`OrderBook`) in einer Collection. `Market` ist verantwortlich, dass ein Auftrag in das richtige Auftragsbuch eingeordnet wird.

6.4.4 Bank

`Bank` hat auch eine Fassadenfunktion. `Bank` verwaltet die Depots (`Depot`) in einer Collection. Ebenso ist `Bank` verantwortlich für den Geld- und Aktientransfer zwischen den Depots.

6.4.5 ObjectManager

Der `ObjectManager` ist eine dynamische Collection, die von `Bank` und `Market` gebraucht wird. Der `ObjectManager` verwaltet die Objekte (z.B. Auftragsbücher oder Depots) in einer Liste. Dabei ordnet der `ObjectManager` jeweils die Objekte, nach ihrem Zugriffsmuster. Der `ObjectManager` ist ein `TransactionListener`. Sobald eine Transaktion abgeschlossen (persistent und konsistent gemacht) wurde, schneidet der `ObjectManager` die Liste bei einer bestimmten, definierbaren Stelle ab und so werden die Objekte, auf die selten oder nicht mehr zugegriffen wird, dem `GarbageCollector` übergeben.

6.4.6 OrderBookFactory / DepotFactory

Diese beiden Klassen implementieren die `IObjectFactory`-Schnittstelle. Über diese Schnittstelle ist der `ObjectManager` in der Lage die Objekte zu instanzieren. Wie der Name schon sagt, ist die `OrderBookFactory` für das Instanzieren für `OrderBook`-Objekte zuständig und die `DepotFactory` für das Instanzieren der Klasse `Depot`.

6.4.7 OrderBook

Die Klasse `OrderBook` ist verantwortlich für die Verwaltung aller Instanzen (Beans) der Klasse `SystemOrder`. Die Objekte verwalten je eine Aktienart (z.B. eine Partei-Aktie bei einer Wahl, JA-Aktie oder NEIN-Aktie bei einer Abstimmung). Eine Instanz der Klasse `OrderBook` ist verantwortlich, dass Käufe und Verkäufe richtig abgewickelt werden und produziert dazu `Match`-Beans. Das Auftragsbuch ist verantwortlich, dass beim Instanzieren die Beans (`SystemOrder`) von der Datenbank geholt werden und es ist auch dafür verantwortlich, dass es die veränderten Daten persistent macht. Dazu implementiert die Klasse die `IPersistentable`-Schnittstelle.

6.4.8 Depot

Die Klasse `Depot` ist für die Verwaltung von Depotdaten zuständig. Dazu braucht es die Beans `UserOrder`, `Money` und `StockAmount`. Wie die Klasse `OrderBook` ist auch `Depot` für das Laden und Speichern der Daten auf die Datenbank zuständig. `Depot` implementiert deshalb auch die `IPersistentable`-Schnittstelle.

6.4.9 ManagedList

Die `ManagedList` ist eine generische Collection, welche in etwa die gleichen Funktionen besitzt, wie eine gewöhnliche `ArrayList` (Fassende). Das Besondere ist aber, dass ein Objekt nur einmal in eine `ManagedList` eingetragen werden kann. Wird nochmals die `Add(object o)`-Methode mit einer längst gespeicherten Referenz aufgerufen, so wird dieser Aufruf von der `ManagedList` einfach ignoriert. Die zweite besondere Funktion einer `ManagedList` ist, dass sie es zulässt, dass man Referenzen während einer `foreach`-Anweisung hinzufügt oder löscht. Dazu iteriert die `ManagedList` während einer `foreach`-Anweisung über eine temporäre Kopie der Liste. (Normale Collections machen Fehler, wenn man sie während dem Iterieren ändert.)

Gebraucht wird die `ManagedList` vom `TransactionController` zum Speichern der veränderten Depots (`dirtyDepots`) und vom `OrderBook` zum Verwalten der `SystemOrder`-Beans und vom `Depot` zur Verwaltung der `UserOrder`- und `StockAmount`-Beans.

6.4.10 ManagedStack

Der `ManagedStack` ist ähnlich wie die `ManagedList` eine Fassade eines normalen Stacks. Ebenso mit der Besonderheit, dass Referenzen nur einmal eingetragen werden dürfen. Gebraucht wird dieser beim `TransactionController` um die `Match`-Beans zu verwalten und beim `TransactionHandler` um die veränderten Objekte (`dirtyObjects`) zu referenzieren.

6.4.11 DbReader

Der `DbReader` ist dafür verantwortlich, dass über SQL-Statements oder über Tabellennamen mit optionaler Angabe von Schlüsseln und Schlüsselwerten Tupels von der Datenbank in Arrays mit Objects umgewandelt und in eine Collection eingetragen werden. Der `DbReader` greift via SQL auf eine SQL-fähige Datenbank zu.

6.4.12 DbTransactionWriter

Der `DbTransactionWriter` ist dafür verantwortlich, dass alle Tupels innerhalb einer Datenbank-Transaktion geschrieben werden. Der `DbTransactionWriter` greift ebenfalls via SQL auf eine SQL-fähige Datenbank zu.

6.4.13 IdCreator

Die Klasse `IdCreator` stellt über ein statisches Property `NewId` eine zeitlich eindeutige ID als Long-Wert zur Verfügung. Der `IdCreator` kann während einer Sekunde 1000 eindeutige IDs produzieren.

6.4.14 Match

`Match` ist ein Bean, welches einen Tupel der Tabelle „Match“ verwaltet. `Match` implementiert die `IPersistentable`-Schnittstelle und kann sich somit selbständig über den `TransactionHandler` persistent machen.

6.4.15 ProcessedOrder

`ProcessedOrder`-Bean verwaltet eine Tupel der Tabelle „ProcessedOrder“, wo `Order`-Beans abgelegt werden, die vom System durch einen Kaufs-/Verkaufsvorgang verändert oder eliminiert wurden. Auch `ProcessedOrder` implementiert die `IPersistenable`-Schnittstelle und kann deshalb in die DirtyList eingetragen werden, um sich später selbst über den `TransactionHandler` persistent zu machen.

6.4.16 Order

`Order` ist eine abstrakte Klasse und implementiert die Gemeinsamkeiten der Beans `UserOrder` und `SystemOrder`. `Order` verwaltet einen Kauf-, bzw. Verkaufsauftrag.

6.4.17 SystemOrder

Ein `SystemOrder`-Bean verwaltet ein Tupel der Tabelle „SystemOrder“. Es wird von den Auftragsbüchern gebraucht.

6.4.18 UserOrder

Ein `UserOrder`-Bean verwaltet ein Tupel der Tabelle „UserOrder“. Es wird von den Depots gebraucht.

6.4.19 Money

Das `Money`-Bean verwaltet ein Tupel der Tabelle „Money“. Es stellt das Geld eines Depots dar.

6.4.20 StockAmount

`StockAmount` verwaltet ein Tupel der Tabelle „StockAmount“. Es stellt die Anzahl Aktien einer bestimmten Sorte dar.

6.4.21 UserOrderFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `UserOrder`.

6.4.22 SystemOrderFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `SystemOrder`.

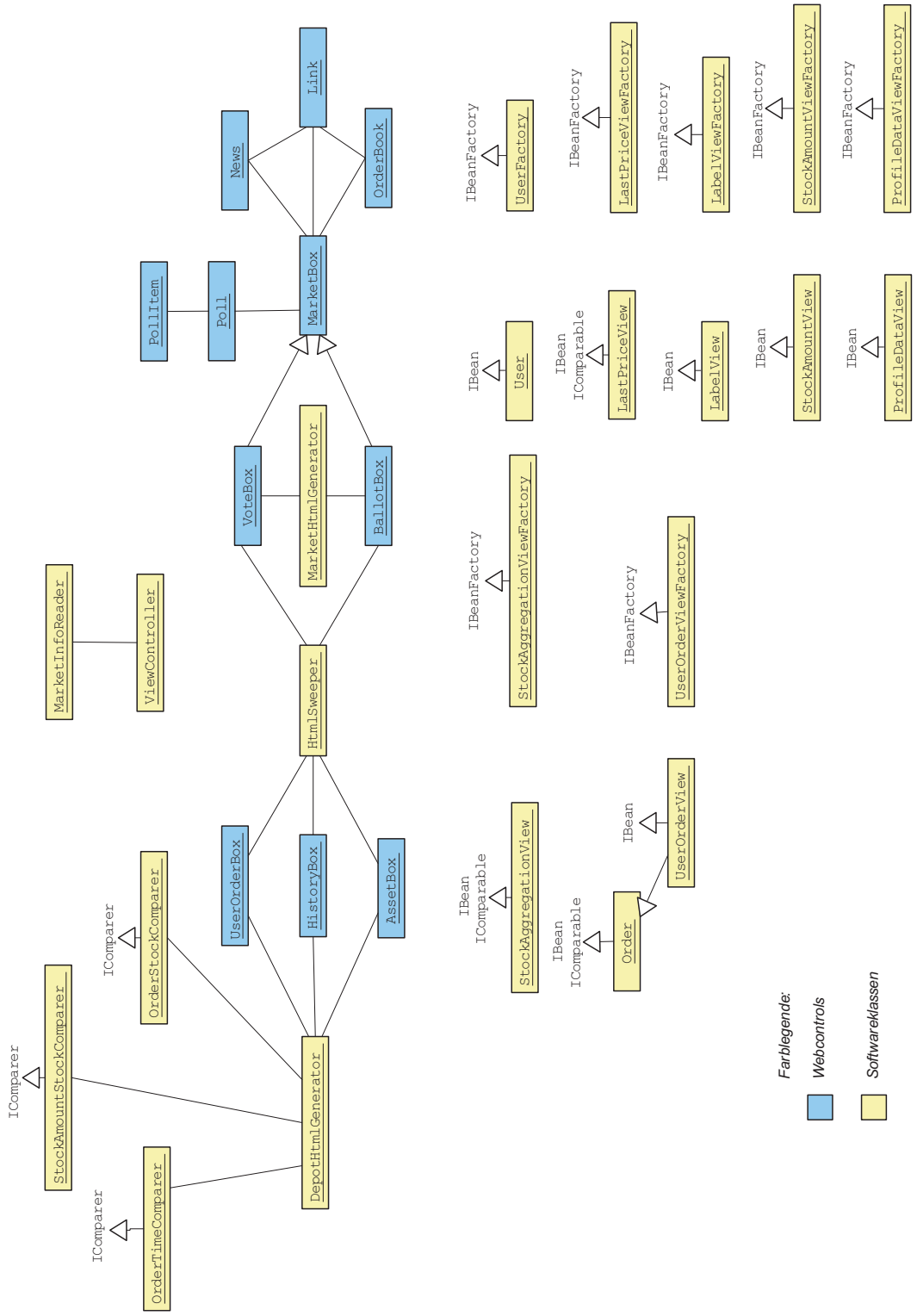
6.4.23 MoneyFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `Money`.

6.4.24 StockAmountFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `StockAmout`.

6.5 View-Klassendiagramm



6.6 Klassen der View

6.6.1 DepotHtmlGenerator

Diese Klasse generiert den HTML-Code für die Depotübersicht. Eine Depotübersicht ist zusammengesetzt aus `AssetBox`, `UserOrderBox` und `HistoryBox`.

6.6.2 HtmlSweeper

Der `HtmlSweeper` stellt den generierten HTML-Code in einer für den Menschen lesbaren Art dar.

6.6.3 MarketHtmlGenerator

Die Klasse MarketHtmlGenerator generiert analog zum DepotHtmlGenerator den HTML-Code auf der Handelsübersichtseite. Eine Handelsübersichtseite ist zusammengesetzt aus mehreren `InfoBox`, `VoteBox` bzw. `BallotBox`.

6.6.4 MarketInfoReader

Der MarketInfoReader liest die Beschreibungen der einzelnen Abstimmungen bzw. Wahlen aus dem marketinfo.xml File. Die Angaben aus dem marketinfo.xml werden für die Beschriftungen auf den Webseiten gebraucht.

6.6.5 OrderStockComparer

Die Klasse `OrderStockComparer` ordnet zwei Orders nach der StockId.

6.6.6 OrderTimeComparer

Die Klasse `OrderTimeComparer` ordnet zwei Orders nach dem Timestamp.

6.6.7 StockAmountStockComparer

Die Klasse `StockAmountStockComparer` ordnet Aktien im Depot nach Aktie (`Stock`).

6.6.8 ViewController

Der ViewController ist die Controller-Klasse für die View. Über diese Controller-Klasse greift die View auf die Business-Logik.

6.6.9 LastPriceView

Das `LastPriceView`-Bean verwaltet ein Tupel der Tabelle „Match“. Es stellt den zuletzt gehandelten Preis einer Aktie zur Verfügung. Die Handelsübersichtseite, die Depotübersicht, wie auch das Auftragseingabefenster brauchen diese Information.

6.6.10 LastPriceViewFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `LastPriceView`.

6.6.11 ProfileDataView

Das `ProfileDataView`-Bean verwaltet ein Tupel der Tabelle „ProfileData“. Die Daten aus dem Anmeldeformular werden über dieses Bean auf die Datenbank geschrieben.

6.6.12 ProfileDataViewFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `ProfileDataView`.

6.6.13 StockAmountView

Das `StockAmount`-Bean verwaltet ein Tupel der Tabelle „StockAmount“. Es wird auf der Depotübersichtseite zum Anzeigen der Aktien im Besitz des Händlers gebraucht.

6.6.14 StockAmountViewFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `StockAmountView`.

6.6.15 SystemOrderAggregationView

Das `SystemOrderAggregationView`-Bean verwaltet ein Tupel der Tabelle „SystemOrder“ und aggregiert gleich die SystemOrders über ein SQL-Statement nach Aktien und Preis. So werden die Aufträge zusammengefasst. Dieses Bean wird auf der Handelsübersichtseite zum Anzeigen der nächsten drei Bid- bzw. Ask-Aufträge gebraucht.

6.6.16 SystemOrderAggregationViewFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `SystemOrderAggregationView`.

6.6.17 User

Das `User`-Bean verwaltet ein Tupel der Tabelle „User“. Es enthält alle nötigen Informationen für die Authentisierung. Meldet sich ein Benutzer neu an, wird ein neues Tupel in die Tabelle User geschrieben.

6.6.18 UserFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `User`.

6.6.19 UserOrderView

Das `UserOrderView`-Bean verwaltet ein Tupel der Tabelle „UserOrder“. Es wird in der Depotübersicht, zum Anzeigen aller hängigen Aufträge gebraucht. Das Bean kann nicht auf die Datenbank geschrieben werden. (read only)

6.6.20 UserOrderViewFactory

Erzeugt über ein ObjectArray eine Instanz der Klasse `UserOrderView`.

6.6.21 AssetBox.

Die Klasse `AssetBox` definiert ein WebControl zum Darstellen des Aktienbestands in der Depotübersicht. Der HTML-Code für die `AssetBox` wird über den `DepotHtmlGenerator` generiert.

6.6.22 BallotBox

Die Klasse `BallotBox` definiert ein WebControl zum Darstellen einer Abstimmung auf der Handelsübersichtseite. Der HTML-Code für die `BallotBox` wird über den `MarketHtmlGenerator` generiert.

6.6.23 HistoryBox

Die Klasse `HistoryBox` definiert ein WebControl zum Darstellen der ausgeführten Aufträge in der Depotübersicht. Der HTML-Code für die `HistoryBox` wird über den `DepotHtmlGenerator` generiert.

6.6.24 Link

Die Klasse `Link` definiert ein WebControl und stellt einen einfachen Link dar.

6.6.25 MarketBox

Ist die MutterBox von VoteBox und BallotBox. Es generiert den Code, der für beide identisch ist. Z.B. eine OrderBook-Zeile.

6.6.26 News

Die Klasse `News` definiert ein WebControl zum Darstellen der News auf der Handelsübersichtseite.

6.6.27 OrderBook

Ist ein Klasse für ein Child-Node der MarketBox.

6.6.28 Poll

Ist ein Klasse für ein Child-Node des OrderBook.

6.6.29 PollItem

Ist ein Klasse für ein Child-Node des Poll.

6.6.30 UserOrderBox

Die Klasse `UserOrderBox` definiert ein WebControl zum Darstellen der hängigen Aufträge in der Depotübersicht. Der HTML-Code für die `UserOrderBox` wird über den `DepotHtmlGenerator` generiert.

6.6.31 VoteBox

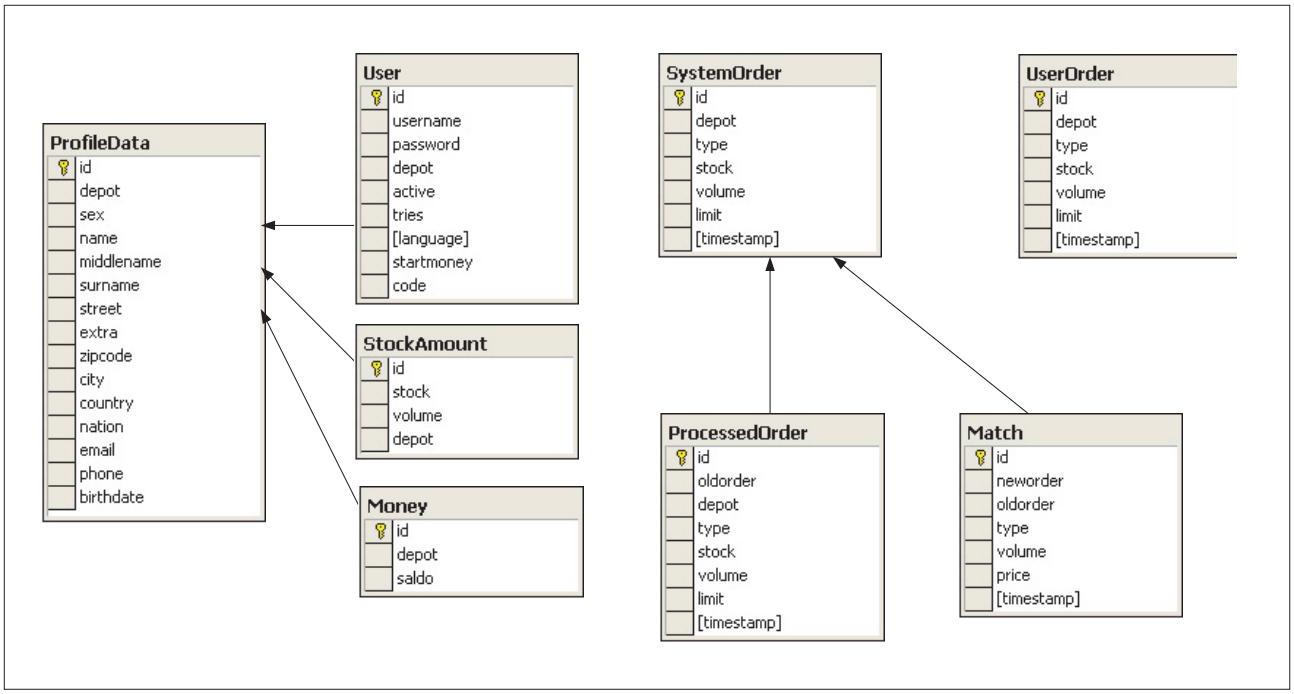
Die Klasse `VoteBox` definiert ein WebControl zum Darstellen einer Wahl auf der Handelsübersichtseite. Der HTML-Code für die `VoteBox` wird über den `MarketHtmlGenerator` generiert.

6.6.32 Validator

Die Klasse Validator validiert die Eingaben aus dem Anmeldeformular auf der Webseite. Die Validierung wird aufgrund regulärer Ausdrücke durchgeführt.

6.7 Der Datenbankaufbau

6.7.1 Das ER-Schema



6.7.2 Beschreibung der Tabellen

Tabellenname: ProfileData
Beschreibung: Registriert sich ein neuer Benutzer werden die Angaben aus dem Anmeldeformular hier gespeichert.

ProfileData

Column Name	Data Type	Length	Allow Nulls
id	bigint	8	
depot	bigint	8	
sex	char	1	
name	varchar	100	
middlename	varchar	100	✓
surname	varchar	100	
street	varchar	100	
extra	varchar	100	✓
zipcode	varchar	100	
city	varchar	100	
country	varchar	100	✓
nation	varchar	100	
email	varchar	100	✓
phone	varchar	100	✓
birthdate	varchar	100	✓

Tabellenname: User

Beschreibung: Neben den Profildaten, werden in der Tabelle User weitere für den Betrieb wichtige Daten festgehalten. Neben Username und Passwort (zum Einloggen) wird auch gleich das Startkapital gespeichert, damit man seine Performance ausrechnen kann.
Meldet sich ein neuer Benutzer an, ist sein Username noch gesperrt. Um das Konto zu aktivieren, haben wir die Spalte "code" angelegt, was nichts anderes als eine einmalige Zufallszahl ist. Aktiviert sich der neue Benutzer mit dem Mail wird der Code automatisch geändert, damit sich der Benutzer nicht mit dem gleichen Code nochmals frei schalten kann.

User

Column Name	Data Type	Length	Allow Nulls
id	bigint	8	
username	varchar	100	
password	varchar	100	
depot	bigint	8	
active	char	1	
tries	int	4	
[language]	varchar	50	
startmoney	float	8	
code	bigint	8	

Tabellenname: Money

Beschreibung: Für jedes Depot werden in dieser Tabelle die aktuellen flüssigen Mittel festgehalten.

Money

Column Name	Data Type	Length	Allow Nulls
id	bigint	8	
depot	bigint	8	
saldo	float	8	

Tabellenname: StockAmount

Beschreibung: Die „StockAmount" Tabelle entspricht der eigentlichen Depotübersicht eines jeden Händlers. Hier ist also jederzeit nachgeführt, was ein Händler gerade an Aktien besitzt.

Money

Column Name	Data Type	Length	Allow Nulls
id	bigint	8	
depot	bigint	8	
saldo	float	8	

Tabellenname: UserOrder
Beschreibung: In dieser Tabelle werden alle User Aufträge gespeichert. Mit den User Aufträgen wird nicht direkt gehandelt. User Aufträge sind wie zwischengespeicherte Aufträge, die zum Augenblick der Eingabe nicht in System Aufträge umgesetzt werden konnten.

UserOrder

Column Name	Data Type	Length	Allow Nulls
id	bigint	8	
depot	bigint	8	
type	char	3	
stock	bigint	8	
volume	bigint	8	
limit	float	8	
[timestamp]	bigint	8	

Tabellenname: SystemOrder
Beschreibung: Hier werden alle System Aufträge gespeichert. Mit den System Aufträgen wird in den Auftragsbüchern gehandelt

SystemOrder

Column Name	Data Type	Length	Allow Nulls
id	bigint	8	
depot	bigint	8	
type	char	3	
stock	bigint	8	
volume	bigint	8	
limit	float	8	
[timestamp]	bigint	8	

Tabellenname: ProcessedOrder
Beschreibung: Machten sich zwei System Aufträge, werden diese aus dem Auftragsbuch gelöscht und im ProcessedOrder archiviert. Aus den Daten aus dieser Tabelle kann man auf die ursprünglichen Aufträge schliessen.

ProcessedOrder

Column Name	Data Type	Length	Allow Nulls
id	bigint	8	
oldorder	bigint	8	
depot	bigint	8	
type	char	3	
stock	bigint	8	
volume	bigint	8	
limit	float	8	
[timestamp]	bigint	8	

Tabellenname: Match

Beschreibung: Gibt's einen Match zwischen zwei System Aufträgen, wird ein neues Tupel in die Tabelle eingefügt. Die Spalte "price" entspricht dem aktuellen Kurs der Aktie und wird zum Zeichnen von Charts benutzt.

Match

Column Name	Data Type	Length	Allow Nulls
id	bigint	8	
neworder	bigint	8	
oldorder	bigint	8	
type	char	3	
volume	int	4	
price	float	8	
[timestamp]	bigint	8	

7 Implementierung

7.1 Sicherheit

Wir haben ein Sicherheitskonzept entworfen. In jeder Schicht und bei jedem Übergang von einer Schicht zu einer anderen in der 3-Tier-Architektur treten Sicherheitsmerkmale auf.

7.1.1 Sicherheit Client

Um dem Client Sicherheit zu bieten, muss man sich bei per Username und Passwort einloggen. Nach drei fehlerhaften Eingaben wird der Account gesperrt. Dies macht es unnötig, dass der Benutzer ein starkes Passwort wählen muss. Da der Benutzer identifiziert bleibt, wird mit cookieless Sessions erreicht, welche ASP.NET zur Verfügung stellt. Bei jeder Seite mit sensiblen Daten oder mit einer sensiblen Aktion wird geprüft, ob der Benutzer noch eingeloggt ist. Ansonsten muss er sich neu einloggen. Die Session-Dauer ist beschränkt. Hat sich z.B. jemand nicht ausgeloogt, kann man nach einer bestimmten Dauer nicht mehr auf die Depotseite zugreifen und keine Aufträge mehr abgeben.

Sicherheitslücken: Die Session kann entführt werden. Da wir aber kein System mit sehr hohen Sicherheitsanforderungen haben, gehen wir davon aus, dass diese Attacke nicht ein echtes Problem ist.

7.1.2 Sicherheit Übergang Client / Business-Tier

Die Sicherheit im Übergang vom Client zum Business-Tier kann mit HTTPS erzeugt werden. In der Testumgebung ist dies nicht der Fall. Es ist aber möglich, dies zu gewährleisten, so dass z.B. das Passwort verschlüsselt übertragen wird. Zwischen Client und Business-Tier sollte eine Firewall eingerichtet werden, die DOS-, SYN- und andere gängige Attacken verhindert.

7.1.3 Sicherheit Business-Tier

In der Testumgebung haben wir ein gehärtetes System. Ebenso setzen wir auf die Sicherheit von .NET und ASP.NET auf. ASPX-Seiten dürfen keine Files schreiben. Die Statistiken werden von einem separaten Prozess erzeugt.

Beim Anmelden wird geprüft, ob die Eingaben Regular Expressions genügen. Der Benutzer muss sich aktivieren. So wird geprüft, ob die angegebene Mailadresse tatsächlich funktioniert.

Benutzer-Accounts können ausgeschaltet werden, so dass ein Händler nicht mehr berechtigt ist zu handeln. Ob ein Benutzer-Account aktiv oder inaktiv ist, wird bei jeder Seite geprüft.

Der ViewController überprüft beim Annehmen von Aufträgen nochmals, ob der Benutzer tatsächlich die Berechtigung hat, diese Aufträge auch abzugeben und ob diese im richtigen Format eintreffen. Es ist deshalb nicht möglich, durch Eingaben über POST-Ereignisse versteckte SQL-Statements abzusetzen.

7.1.4 Sicherheit Übergang Business-Tier / Datenbank-Tier

Da das Business-Tier und das Datenbank-Tier über Socket-Verbindungen die Daten austauschen, kann dieser Übergang nochmals mit einer Firewall geschützt werden. Es werden ausschliesslich SQL-Verbindungen eingesetzt.

7.1.5 Sicherheit Datenbank-Tier

Man kann mit RAID arbeiten. Ebenso ist möglich, ein redundantes Server-System einzurichten. Die Business-Logik schreibt Daten immer in einer Transaktion. Die Daten sind dadurch immer in einem konsistenten Zustand.

7.2 Web-Controls

Wir haben Controls für die Handelsseite und für die Depotseite entworfen.

Folgende Controls existieren: BallotBox, VoteBox, AssetBox, UserOrderBox, HistoryBox

Die BallotBox ist für Abstimmungen gedacht und die VoteBox für Wahlen. Sie werden mit diesen Tags deklariert:

```
<market:BallotBox id="box" runat="server" language="german">
<market:VoteBox id="box" runat="server" language="german">
```

Mit dem Attribut „language“ kann die Sprache gesetzt werden.

Folgende Child-Nodes kommen vor:

```
<img src="pics/genmoratoriumsinitative.gif" />
<OrderBook place="yes" bind="11">
<Related place="left" name="Stammzellen- forschung 2004">
<InfoBar>Handelsschluss: 27. November 2005 - 12:00</InfoBar>
<Poll place="right" values="6" name="07.10.05">
<News place="1" ></News>
  <MoreNews></MoreNews>
```

- Mit dem Tag <Img> wird das Bild festgesetzt.
- Mit dem Tag <InfoBar> kann die Headline gesetzt werden.
- Mit dem Tag <OrderBook> wird eine Auftragsbuch-Zeile generiert. Das Attribut „bind“ beschreibt die OrderBookId auf der Datenbank. Das Tag hat noch Child-Nodes und zwar <BuyLink> und <SellLink> womit man einen Link setzten kann beim „kaufen“- bzw. „verkaufen“-Knopf.
- Mit dem Tag <Poll> kann eine Umfrage hineingehängt werden. Das Tag hat Child-Nodes: Bei BallotBox kann man entweder über <Yes><No><Undef> oder sonst über das Tag <Row> mit Attribut „Place“ Umfrage-Werte einfügen.
- Mit dem Tag <Related> wird eine vergangene Umfrage hineingehängt. Es hat dieselben Child-Nodes wie <Poll>.
- Das Tag <News> hat ein Child-Node <Link> womit man Titel und Link einer Neuigkeit setzten kann. Mit dem Attribut „place“ definiert man wieder den Aufzählungsort.
- Das Tag <MoreNews> definiert den Link auf die Seite, wo mehr News zu finden sind.

Die Controls für die Depotseite sind sehr einfach gehalten:

```
<market:AssetBox runat="server" id="AssetBox1" />
<market:UserOrderBox runat="server" id="UserOrderBox1"
MaxRows="10" />
<market:HistoryBox runat="server" id="HistoryBox1" MaxRows="10"/>
```

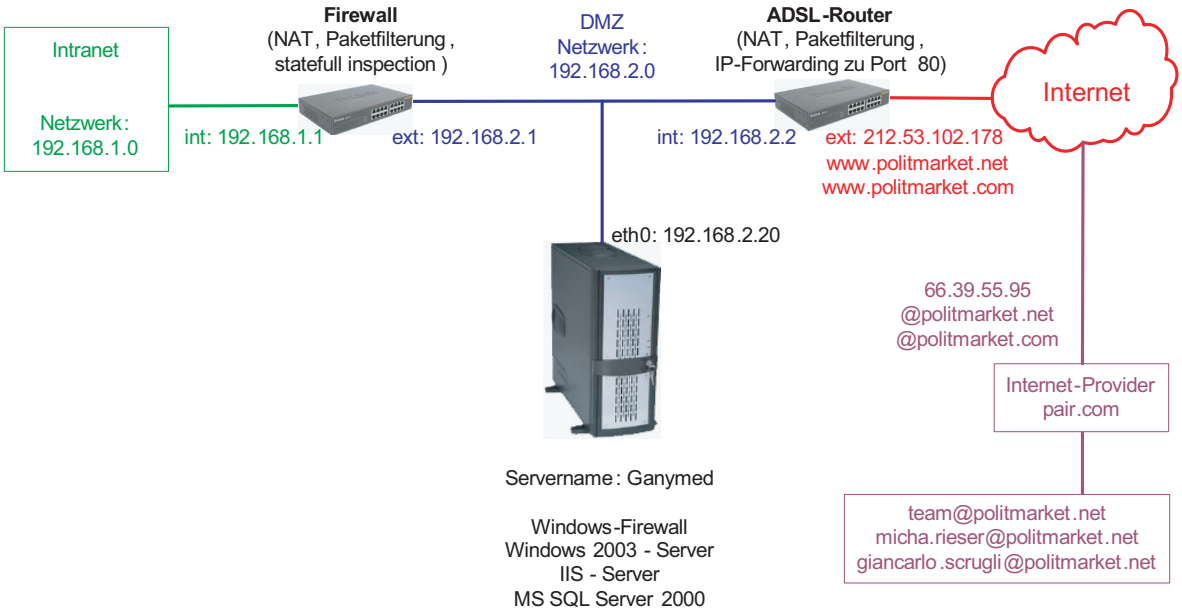
Das Tag <AssetBox> generiert eine Aktienübersicht.
Das Tag <UserOrderBox> generiert eine Übersicht über alle hängigen Aufträgen.
Das Tag <HistoryBox> generiert eine Übersicht über die Kontobewegungen.

Bei allen Tags kann man über das Attribut „language" die Sprache setzten und mit dem Attribut „username" den Benutzer. UserOrderBox und HistoryBox haben noch das Attribut „MaxRows" womit man festlegt, ab wie viel angezeigten Zeilen auf weitere Seiten umgebrochen werden muss.

7.3 Testbetrieb

7.3.1 Die Umgebung

Für den Testbetrieb, war es nahe liegend die Entwicklungsumgebung mit wenigen Anpassungen zu nutzen. Wir haben den Datenbankserver mit dem IIS-Webserver erweitert und in eine DMZ getan. Doch bevor wir den Server ans Netz hängen konnten, mussten wir den Server härten. Wir härteten den IIS, den MS SQL wie auch Windows 2003-Betriebssystem. Mehr dazu im Kapitel 7.3.5.



7.3.2 Spezifikationen

Datenbank- und Webserver:	Betriebssystem:	Windows 2003 Server Enterprise Edition
	Rechnername:	Ganymed
	IP:	212.53.102.178 (statisch)
[nicht vorhanden!!]	Subnet:	255.255.255.0
	CPU:	Pentium IV Sockel 478, Intel, 1024KB Cache, 2,4 GHz
	Mainboard:	Asus P4S800-MX SiS 661 FX Socket 478 MATX
	Power Supply:	300W
	RAM:	480 MB
	Harddisk:	Hitachi Deskstar 7k250, 250 GB
	Harddisk Partitionen:	C(System): 20 GBD(Daten):230 GB
	CDROM:	Plextor CD-R PX-W2410A
	Ethernet:	SIS 900-basierte PCI-Fast Ethernet-Adapter
	Hardware-Adresse:	00:11:D8:E6:01:50
	Grafik:	SiS 661 FX (On Board)
	Dienste:	IIS, MSSQL

7.3.4 Benutzer und Passwörter

	Username	Passwort
Windows 2003 Enterprise Server (IIS):	Politmarket	Pr0gn0s3
MSSQL Server:	marketconnection sa	DAzhw2005 system

7.3.5 Härtung der Systeme

Bevor wir den Webserver in Netz gestellt haben, haben wir darauf geachtet, dass der Server auf Angriffe vom Internet soweit als möglich geschützt ist. Dafür mussten wir sowohl das Betriebsystem, wie auch den Webserver und die Datenbank berücksichtigen. Wir gehen nicht davon aus, dass das System auf jegliche Arten von Angriffen geschützt ist, wir denken aber, dass der Testbetrieb mit diesen Massnahmen ausreichend gesichert wurde.

Die Massnahmen stammen aus einer Checkliste aus dem Microsoft Developer Network (MSDN), zum Sichern eines Web Servers. Allerdings ist diese Checkliste auf Januar 2004 datiert. Wir haben leider keine aktuellere Version der Checkliste aus vertrauenswürdigen Quellen gefunden.

Im Gegensatz dazu wird auf dieser Webseite das aktuelle Gratis Tool Microsoft Baseline Security Analyzer (MBSA) Version 2.0 angeboten. Der MBSA überprüft das System auf die gängigsten Sicherheitslücken. In unserem Fall hat der MBSA gleich den IIS und die MS SQL Datenbank überprüft.

Wir wollen nun die detaillierten Massnahmen zusammentragen.

Windows 2003 Server
- Automatisches herunterladen der Windowsupdates ist aktiviert
- Die Windows Firewall ist aktiviert. (Nur der Port 80 ist offen)
- Der Server befindet sich hinter einem ADSL NAT Router in der DMZ.
- Unnötige Dienste sind deaktiviert
- Der Telnet, FTP und NNTP Dienste sind deaktiviert
- Der TCP/IP Stack wurde nach Anleitung in der Registry auf Denial of Service Attacken gehärtet.
- Nicht benötigte Benutzerkontos sind gelöscht.
- Das Windows Gast Konto ist deaktiviert.
- Das Administrator Konto wurde umbenannt und hat ein starkes Passwort.
- Das Filesystem ist NTFS.
- Alle unnötigen Freigaben sind entfernt.
- Die Freigaben "C$" und "Admin$" sind entfernt.
- Die Policies wurden soweit angepasst, dass eine fehlgeschlagene Anmeldung am System jetzt geloggt wird.
- Auf dem Internet Explorer ist die Sicherheitszone auf „hoch" eingestellt.
- Wir haben uns am Microsoft Security Notification Service angemeldet und werden per E-Mail über aktuelle Sicherheitshinweise informiert.

IIS
- Die Applikation für die Remote IIS Administration unter \Winnt\System32\Inetsrv\IISAdmin ist entfernt.
- Die Resource-Kit-Tools, Utilities und das SDK sind entfernt.
- Beispielapplikationen unter \WINNT\Help\IISHelp und \Inetpub\IISSamples sind entfernt.
- Die Webseite wurde auf eine nicht System-Partition angelegt.
- Potenzielle Virtuelle Verzeichnisse, wie "IISSamples", "IISAdmin", "IISHelp" und "Scripts" sind entfernt.
- Der anonyme Zugriff auf ein virtuelles Verzeichnis hat kein Schreibrecht.
- URLScan ist installiert und konfiguriert.
- IIS Protokollierung ist aktiviert.

MS SQL Server
- Nur ein Benutzer ist Mitglied der "sysadmin"-Rolle, nämlich der Benutzer "sa".
- Das Gastkonto ist auf den Datenbanken "Northwind" und "pubs" wurde gelöscht.

8 Test

8.1 Blackbox-Test OrderBook

8.1.1 Testcase 1

Precondition: Empty OrderBook

Nr.	Action	Result	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=100, limit=100)	- Ord1(id=0) erased, - Matching(ask=0,bid=-1,vol=100,price=100)	

8.1.2 Testcase 2

Precondition: Empty OrderBook

Nr.	Action	Result	
1	Ord1(bid, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(ask, vol=100, limit=100)	- Ord1(id=0) erased, - Matching(ask=-1,bid=0,vol=100,price=100)	

8.1.3 Testcase 3

Precondition: Empty OrderBook

Nr.	Action	Result	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=150, limit=100)	- Ord1(id=0) erased, - Matching(ask=0,bid=-1,vol=100,price=100) - Ord2 (1,bid,vol=50,limit=100) added	

8.1.4 Testcase 4

Precondition: Empty OrderBook

Nr.	**Action**	**Result**	
1	Ord1(bid, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(ask, vol=150, limit=100)	- Ord1(id=0) erased, - Matching(ask=-1,bid=0,vol=100,price=100) - Ord2 (id=1,ask,vol=50,limit=100) added	

8.1.5 Testcase 5

Precondition: Empty OrderBook

Nr.	**Action**	**Result**	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(ask, vol=100, limit=100)	- Ord2(id=1) added	
3	Ord3(bid, vol=200, limit=100)	- Ord1(id=0) erased, - Ord2(id=1) erased - Matching(ask=0,bid=-1,vol=100,price=100) - Matching(ask=1,bid=-1,vol=100,price=100)	

8.1.6 Testcase 6

Precondition: Empty OrderBook

Nr.	**Action**	**Result**	
1	Ord1(bid, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=100, limit=100)	- Ord2(id=1) added	
3	Ord3(ask, vol=200, limit=100)	- Ord1(id=0) erased, - Ord2(id=1) erased - Matching(ask=-1,bid=0,vol=100,price=100) - Matching(ask=-1,bid=1,vol=100,price=100)	

8.1.7 Testcase 7

Precondition: Empty OrderBook

Nr.	**Action**	**Result**	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=100, limit=50)	- Ord1(id=0) erased - M atching(ask=0,bid=-1,vol=100,price=100)	

8.1.8 Testcase 8

Precondition: Empty OrderBook

Nr.	**Action**	**Result**	
1	Ord1(bid, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(ask, vol=100, limit=150)	- Ord1(id=0) erased - Matching(ask=-1,bid=0,vol=100,price=100)	

8.1.9 Testcase 9

Precondition: Empty OrderBook

Nr.	Action	**Result**	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=100, limit=150)	- Ord2(id=1) added	

8.1.10 Testcase 10

Precondition: Empty OrderBook

Nr.	Action	**Result**	
1	Ord1(bid, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(ask, vol=100, limit=50)	- Ord2(id=1) added	

8.1.11 Testcase 11

Precondition: Empty OrderBook

Nr.	Action	Result	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=100, limit=50)	- Ord1(id=0) erased - Matching(ask=0,bid=-1,vol=100,price=100)	

8.1.12 Testcase 12

Precondition: Empty OrderBook

Nr.	Action	Result	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(ask, vol=200, limit=100)	- Ord2(id=1) added	
3	Ord3(bid, vol=150,limit=100)	- Ord1(id=0) erased, - Ord2(id=1) erased - Ord2(id=2,ask,vol=150,limit=100) added - Matching(ask=0,bid=-1,vol=100,price=100) - Matching(ask=1,bid=-1,vol=50,price=100)	

8.1.13 Testcase 13

Precondition: Empty OrderBook

Nr.	Action	Result	
1	Ord1(bid, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=200, limit=100)	- Ord2(id=1) added	
3	Ord3(ask, vol=150,limit=100)	- Ord1(id=0) erased, - Ord2(id=1) erased - Ord2(id=2,bid,vol=150,limit=100) added - Matching ask=-1,bid=0,vol=100,price=100) - Matching ask=-1,bid=1,vol=50,price=100)	

8.1.14 Testcase 14

Precondition: Empty OrderBook

Nr.	Action	Result	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(ask, vol=200, limit=100)	- Ord2(id=1) added	
3	Ord3(bid, vol=150,limit=50)	- Ord1(id=0) erased, - Ord2(id=1) erased - Ord2(id=2,ask,vol=150,limit=100) added - Matching(ask=0,bid=-1,vol=100,price=100) - Matching(ask=1,bid=-1,vol=50,price=100)	

8.1.15 Testcase 15

Precondition: Empty OrderBook

Nr.	Action	Result	
1	Ord1(bid, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=200, limit=100)	- Ord2(id=1) added	
3	Ord3(ask, vol=150,limit=150)	- Ord1(id=0) erased, - Ord2(id=1) erased - Ord2(id=2,bid,vol=150,limit=100) added - Matching ask=-1,bid=0,vol=100,price=100) - Matching ask=-1,bid=1,vol=50,price=100)	

8.1.16 Testcase 16

Precondition: Empty OrderBook

Nr.	Action	Result	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(ask, vol=200, limit=100)	- Ord2(id=1) added	
3	Ord3(bid, vol=150,limit=150)	- Ord3(id=2) added	

8.1.17 Testcase 17

Precondition: Empty OrderBook

Nr.	**Action**	**Result**	
1	Ord1(bid, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=200, limit=100)	- Ord2(id=1) added	
3	Ord3(ask, vol=150,limit=50)	- Ord3(id=2) added	

8.1.18 Testcase 18

Precondition: Empty OrderBook

Nr.	**Action**	**Result**	
1	Ord1(ask, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(ask, vol=200, limit=75)	- Ord2(id=1) added	
3	Ord3(bid, vol=150,limit=50)	- Ord1(id=0) erased, - Ord2(id=1) erased - Ord2(id=2,ask,vol=150,limit=75) added - Matching(ask=0,bid=-1,vol=100,price=100) - Matching(ask=1,bid=-1,vol=50,price=75)	

8.1.19 Testcase 19

Precondition: Empty OrderBook

Nr.	Action	**Result**	
1	Ord1(bid, vol=100, limit=100)	- Ord1(id=0) added	
2	Ord2(bid, vol=200, limit=125)	- Ord2(id=1) added	
3	Ord3(ask, vol=150,limit=150)	- Ord1(id=0) erased, - Ord2(id=1) erased - Ord2(id=2,bid,vol=150,limit=125) added - Matching(ask=-1,bid=0,vol=100,price=100) - Matching(ask=-1,bid=1,vol=50,price=125)	

9 Quellenverzeichnis

Amt für Umweltschutz, Kanton Zug: Flechten und Luftqualität im Kanton Zug: Wirkungskontrolle 2003, Zug, 9. Juli 2004

Iowa Electronic Markets, http://www.biz.uiowa.edu/iem/, 27. Oktober 2005

Feldmeier, Herrmann: Warnung vor gefährlichen Insektenplagen, in: Forschung und Technik, NZZ, 19. Oktober 2005.

Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John: Entwurfsmuster, Addison-Wesley Verlag, München (1996)

Larman, Craig: UML 2 und Patterns angewendet – Objektorientierte Softwareenticklung, mtip-Verlag, Bonn (2005)

Berr Wolfgang, Birngruber Dietrich, Mössenböck Hanspeter, Wöss Albrecht: Die .NET-Technologie, dpunkt.verlag, Heidelberg (2003)

Kofler, Michael und Eller, Frank: Visual C# Grundlagen, Programmiertechniken, Windows-Programmierung, Addison-Wesley Verlag, München (2005)

Schwichtenberg, Holger und Eller, Frank: Programmierung mit der .NET Klassenbibliothek, Addison-Wesley Verlag, München (2004)

Lorenz, Patrick A.: Grundlagen und Profiwissen ASP.NET Webserverprogrammierung und XML Web Services im .NET-Framework, Carl Hanser Verlag, München (2004)

Rieser, Micha und Scrugli, Giancarlo: Börsensimulation, Projektarbeit 2, Zürcher Hochschule Winterthur (2005)

10 Anhang

10.1 Detaillierter Iterationsplan

	Wo 1 5.9.-11.9.	Wo 2 12.9.-18.9.	Wo 3 19.9.-25.9.	Wo 4 26.9.-2.10.	Wo 5 3.10.-9.10.	Wo 6 10.10.-16.10.	Wo 7 17.10.-23.10.	Wo 8 24.10.-30.10.
Mo	Analyse Views als Webpages und Use Case handelnAktie. Domain Model. Use Case Diagram 1. Version	KNABEN-SCHIESSEN	Datenbank-Anbindung mit OrderBooks und Depots	vollständig Recoding	Recoding + Redesign	Implementierung WebControl Handel-Seite	Implementierung “Statistik”	Dokumentation
Di	Design Use Case handelnAktie	Design Depot und Entscheidung für komlexes System für mehrere Freiheiten für Benutzer	Code Review + Redesign	vollständig Recoding	Abschliessen Recoding + Redesign	Implementierung WebControl Depot-Seite	Verbesserung Usability (AddOrder, Login/Logout)	Dokumentation Korrektur
Mi	Implementierung handelnAktie - Datenbank Zugriff Access, DbServices	Implementierung Depot + TestOverView	Implementierung erste “Gehversuche” mit ASP.NET	Recoding + Test	Start ASP.Net! Erstellen der Handelsseite (mit Repeater) + Login Maske	Entwicklung an der Webseite	“Härten” des Servers Aufschalten des “Testbetriebs” ->Politmarket.net ist “live”!	Korrektur
Do	Implementierung handelnAktie - OrderBook und DbServices	Implementierung Depot	Implementierung erste “Gehversuche” mit ASP.NET	Recoding + Redesign	Erstellen der Handelsseite (ohne Repeater) restliche Seite des Webportals sind erstellt Navigation	Entwicklung an der Webseite	Dokumentation Inhaltsverzeichnis erstellt	Korrektur Drucken
Fr	Besprechung 14:00 + Test OrderBook (Blackbox und Whitebox)	Besprechung 13:30 + wenige Tests	Besprechung 14:00 / Vorbereitung Redesign und Recoding	Besprechung 14:00 / Vorbereitung Redesign und Recoding	Besprechung 14:00 Implementierung Webcontrols Handelsseite Anmeldeformular	Besprechung 14:00 Anpassungen	Besprechung 14:00 Dokumentation	Binden Abgabe der DA bei der Ierax in Zürich
Sa	Zusammenbau und Aufsetzen neuer Server “ganymed” inkl. Installation MS-SQL deutsch. -> Probleme: Keine Service-Packs verfügbar	Datenbankserver: Installation MS-SQL english + sp1 + sp2 + sp3	Weak References untersucht			Validieren der Eingaben über Regular Expressions	Dokumentation	
So		Einrichtung Datenbank mit Tabellen: userorders, systemorders, money, depot				Analyse und Design “Statistik”	Dokumentation	

10.2 Glossary

A

AddOrder(stock,price,volume,type)	Ein neues Auftrag-Bean wird zum Verarbeiten ans System übergeben.
ask	Entspricht einem Kaufauftrag; Nachfrage
Ask_Stock(DepotNumber)	Wird ausgelöst, wenn der Händler über die Depotübersicht eine Aktie kaufen möchte. Es wird gleich die Depotnummer des Händlers mitgegeben.

B

Bank	Die Klasse verwaltet alle Depots.
Bean	Entspricht einem Tupel aus der Datenbank. Ein Bean ist ein Objekt mit bestimmten Feldern.
bid	Entspricht einem Verkaufsauftrag; Angebot
Bid_Stock(stock)	Wird ausgelöst, wenn der Händler über die Depotübersicht eine Aktie verkaufen möchte. Beim Verkauf ist die Aktie schon bekannt.

C

Cancel()	Schliesst das offene Fenster.
Click_Depot()	Öffnet die Depotübersicht.
Click_delete_Order(order)	Wird ausgelöst, wenn man ein Auftrag aus der Depotübersicht löschen möchte. Es wird auch gleich das betreffende Auftrag-Bean «order» mitgegeben.
CommitOrder()	Bestätigt Auftrag.

D

DBMS	Daten Bank Management System; Bespiel Oracle, MS SQL, DB2.
DbReader	Stellt über ADO.NET eine Verbindung zur Datenbank her und liest Tupel aus der Datenbank.
DbTransactionWriter	Stellt über ADO.NET eine Verbindung zur Datenbank her und schreibt Tupel in die Datenbank.
Delete_Order(order)	Löscht das Auftrag-Bean «order».
Depot	Analog zum Depot im Realen; hier werden also die Aktien im Besitzt des Händlers zusammengetragen. Neben den Aktien werden im Depot auch die flüssigen Mittel nachgeführt.
DepotFactory	Factory zum erstellen eines Depot-Objekts. (Factory-Pattern)

E

Error(string)	Meldet eine Fehlermeldung mit Text als String.

F

Factory-Pattern	Entwurfsmuster aus der Softwarentwicklung. In einer Factory werden Objekte eines bestimmten Typs «produziert».

G

H

I

IBean	Schnittstelle über das man auf ein Bean zugreifen kann.
ICollection	Standardschnittstelle einer Collection.
IComparable	Schnittstelle für das Vergleichen zweier Objekte.
IDeletable	Schnittstelle zum Löschen aller Beans vom Depot und OrderBook
IDelete	Schnittstelle zum Löschen von Beans.
IDirty	Schnittstelle zum Eintragen in die Dirtylist.

IEnumerable	Schnittstelle zum Iterieren über eine Collection.
IInit	Schnittstelle zum Erstellen von Beans.
IObjectFactory	Schnittstelle zum erstellen von Depot- und Orderbook-Instanzen.
IPersistentable	Schnittstelle zum Persistentmachen von Beans.
ISave	Schnittstelle zum Speichern von Beans.
IIS	Internet Information Server; Webserver von Microsoft
IStack	Schnittstelle für den ManagedStack.
ITransaction	Zusammenfassung der Schnittstellen IDirty, IInit, IDelete, ISave und zusätzlichem Eintragen vom TrasactionListener.
ITransactionListener	Schnittstelle um Eröffnung und Abschluss einer Transaktion zu verfolgen.

J

K

L

M

ManagedList	
Market	Die Klasse verwaltet alle OrderBooks.
Money	Bean zum Verwalten der flüssigen Mittel auf der Datenbank.
MoneyFactory	Factory zum Erstellen eines Money-Objekts. (Factory-Pattern)
MS SQL	Microsoft Sturctured Query Language; Datenbankserver von Microsoft.

N

NewOrder(type,stockId)	Stellt einen neuen Auftrag vom Typ «type» und der Aktie mit der Id «stockId».

O

ObjectManager	
Order	Bean Auftrag
Orderbook	Auftragsbuch
OrderbookFactory	Klasse zum Erstellen eine Autragbuch-Objekt.

P

ProcessedOrder	

Q

R

S

Show_Depot()	Erstellt auf der Ansicht eine neue Depotübersicht.
Show_Order(order)	Zeigt das Autrag-Objekt «order» an.
ShowErrorForm()	Zeigt auf der Eingabemaske eine Fehlermeldung an.
ShowOrderForm()	Zeigt Eingabemaske für einen neuen Auftrag an.
ShowOrderWindow()	Öffnet Fenster mit Auftrag.
Show_OrderMask(type)	Zeigt das Auftragsformular an vom Typ «type». Der Typ ist entweder «ask» oder «bid».
Show_OrderMask(stock,bid)	Analog zu Show_OrderMask(type). Hier wird zusätzlich noch die Aktie über eine Id angegeben.
Stock	Aktie
SystemOrder	System Auftrag. System Aufträge werden im Auftragsbuch gehandelt.
SystemOrderFactory	Die Factory zum Erstellen eines System Auftrag Objekt.

T

TransactionController	Ist verantwortlich für die Transaktionssteuerung in der Business-Logik.

TransactionHandler	Stellt Beans für die Verarbeitung in der Business-Logik zur Verfügung. Weiter stellt es über der DbReader und DbTransactionWriter eine Verbindung zu Datenbank zur Verfügung.
TransmitData(type, stockID, quantity, limit)	
U	
UserOrder	Entspricht dem eingegebenen Auftrag des Händlers. Ein UserOrder wird anhand der Informationen aus dem Depot des händlers in ein SystemOrder umgewandelt. UserOrders werden nicht in Auftragsbüchern gehandelt.
UserOrderFactory	Factory zum erstellen eines UserOrder-Objekts. (Factory-Pattern)
V	
W	
X	
Y	
Z	

10.3 Benutzerhandbuch

Dieses Benutzerhandbuch wird so auf der Website unter „Spielregeln“ veröffentlicht.

10.3.1 Um was geht es?

PolitMarket ist ein Spiel. Es hat aber auch einen ernsten Hintergrund. (So toternst ist er aber auch wieder nicht.) Auf diesem System können Sie als Händler Aktien kaufen und verkaufen. Es handelt sich aber nicht um echte Aktien, sondern um fiktive Aktien eines Wahl- bzw. Abstimmungsausgangs. Die Aktien funktionieren ähnlich wie Optionen an einer Börse. Der Schlusskurs der Aktien entspricht dem Abstimmungs- bzw. Wahlergebnis in Prozent am Abstimmungssonntag. Beispiel: Wenn Sie SP-, SVP- und Grüne-Aktien besitzen, dann sind diese bei der Nationalratswahl genau soviel wert, wie die Parteien prozentual Stimmen machen. Macht also die SP 25%, die SVP 30% und Grüne 10%, dann ist die SP-Aktie 25 Spielfranken, die SVP-Aktie 30 Spielfranken und natürlich die Grüne-Aktie 10 Spielfranken wert. Es ist also wichtig, dass Sie möglichst genau den Wahlausgang „vorhersehen“. Sie können diese Fähigkeit in einen ökonomischen Gewinn (in Spielfranken) ummünzen. Sobald Aktien zu tief angeboten werden, können Sie z.B. (und um beim Beispiel zu bleiben) SVP für 25 Spielfranken kaufen und sich beim Wahlausgang die 30 Spielfranken auszahlen lassen. Das macht +5 Spielfranken Gewinn. Sie können natürlich jederzeit auch Verkaufsaufträge von SVP-Aktien von 35 Spielfranken aufgeben. Wenn diese Ihnen jemand abkauft, haben Sie auch einen Gewinn von +5 Spielfranken erzielt.

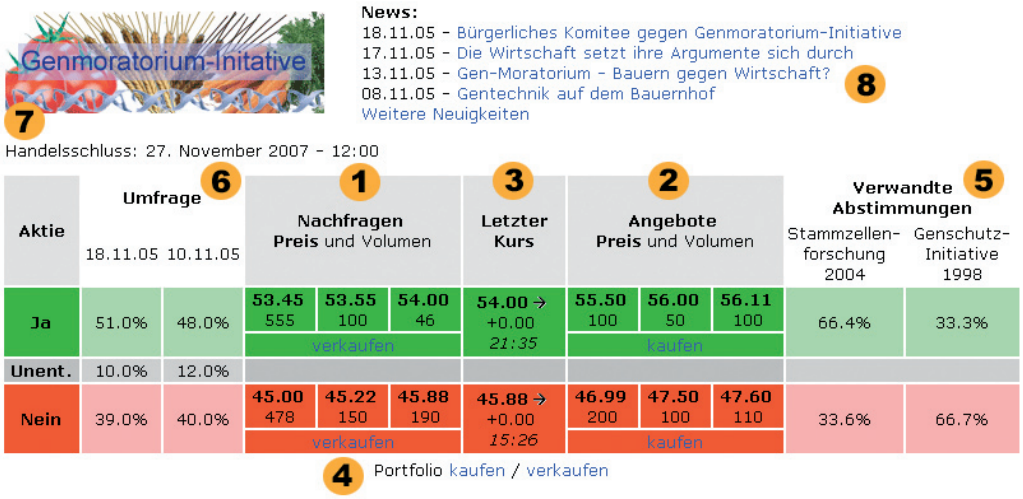
Der ernste Hintergrund dabei ist, dass je mehr Händler versuchen diese Aktien richtig einzuschätzen, sich der Kurs als eine Prognose für den Abstimmungs- und Wahlausgang eignet. Es ist deshalb sehr wichtig, dass Sie dieses Spiel genau deshalb ernst nehmen und vesuchen sich im Freundeskreis, über Medien und über diese Plattform zu informieren, wie realistisch ein Kurs momentan ist und dann auch die richtigen Entscheide treffen. Machen Sie das gut, erzielen sie dabei einen Gewinn. Machen sie das weniger gut, na dann treten Sie auf der Stelle oder machen sogar Verlust (was wir natürlich nicht hoffen).

10.3.2 Beginn

Nachdem Sie sich erfolgreich angemeldet und Ihr Konto aktiviert haben, können Sie bereits handeln. Sie bekommen am Anfang 40‘000 Spielfranken und pro Aktie 200 Stück. Diese Aktien haben einen Wert von 60‘000 Spielfranken. (Aktien können in Portfolios zusammengefasst werden. Ein Portfolio besteht aus einer Ja- und einer Nein-Aktie bei einer Abstimmung oder aus je einer Parteiaktie bei einer Wahl. Ein Portfolio ist immer und jederzeit 100 Spielfranken wert und kann über die Bank gekauft oder verkauft werden!) Sie bekommen also ein Startguthaben von 100‘000 Spielfranken. Sollten Sie beim ersten Aufsuchen Ihrer Depotübersicht bemerken, dass Sie bereits Gewinn oder Verlust gemacht haben, so liegt das daran, dass der Markt ein Portfolio nicht exakt zu 100 Spielfranken bewertet. Das macht aber nichts, da dies ein ständiges Pendeln um die 100 Spielfranken herum ist.

10.3.3 Handelsübersichtseite

Hier sehen sie, welche Möglichkeiten Sie auf der Handelseite haben:



1 Hier stehen fett geschrieben die drei höchsten Nachfragepreise, die im Auftragsbuch zu finden sind. Unten dran sind die Volumen aller Aufträge, die denselben Nachfragepreis haben. Beim Link «verkaufen» öffnet sich ein Auftragsfenster, wo sie einen Verkauftrag platzieren können. Ob Sie dann wirklich sofort verkaufen, hängt davon ab, ob sie eine genug tiefe Limite eingegeben haben.

2 Wie bei den Nachfragen stehen hier fett geschrieben die drei tiefsten Angebotspreise das Auftragsbuchs. Ebenso sind unten dran die Volumen aller Aufträge zu finden, die denselben Angebotspreis tragen. Beim Link «kaufen» öffnet sich ein Auftragsfenster, wo sie einen Kaufauftrag platzieren können.

3 Hier sehen Sie den letzten Kurs mit dem Volumen und der Zeit, wo der Handel über die Bühne ging. Ebenso zeigt Ihnen der Pfeil die Tendenz an, wie sich der Kurs seit Mitternacht (0:00 Uhr) entwickelt hat.

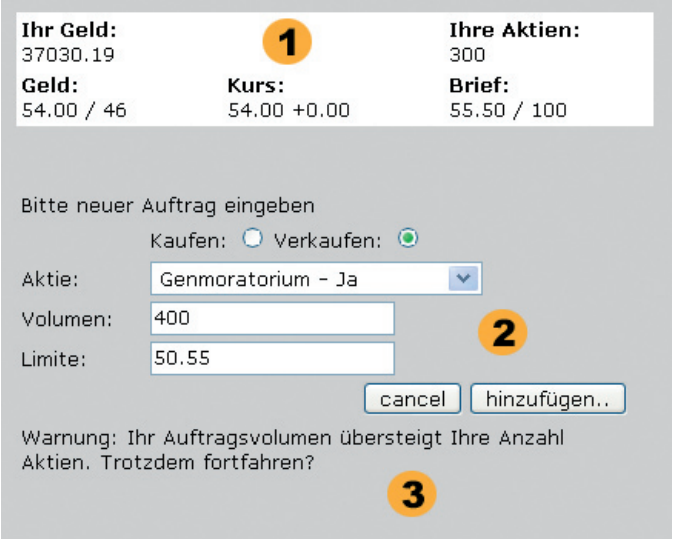
4 Hier können Sie Portfolios mit der Bank handeln. Ein Portfolio ist immer je eine Aktie einer Börse. In diesem Beispiel ist ein Portfolio also eine JA-Aktie und eine Nein-Aktie. Diese beide Aktien zusammen bilden ein Portfolio, welches genau 100 Spielfranken wert ist. Wichtig: Der Handel geht immer über die Bank und nicht über die Börse! Wenn Sie also bemerken, dass z.B. die Nachfragen zusammen über 100 Spielfranken steigen, dann lohnt es sich, über die Bank ein Portfolio zu kaufen und dieses am Markt wieder über normale Verkaufsaufträge zu verkaufen.

5 Hier werden Ihnen Ergebnisse verwandter Abstimmungen gezeigt. Diese sollen Ihnen helfen, sich zu orientieren, wie realistisch der Kurs einer Aktie der gehandelten Vorlage ist.

6 Hier werden Ihnen Ergebnisse neuster Umfragen präsentiert. Wie bei den verwandten Abstimmungen sollen Ihnen diese Umfragen helfen, sich ein Bild zu machen, wie hoch der Preis am Ende tatsächlich ausfallen wird.

7 Hier sehen Sie den Zeitpunkt des Handelsschlusses. Der Handelsschluss ist bei Schliessung der Urnen. Ab diesem Zeitpunkt wird der Handel der Aktien ausgesetzt. Sobald das Schlussergebnis der Wahl oder Abstimmung feststeht, werden die verbleibenden Aktien in Ihrem Depot mit Hilfe des Wahlergebnis in Spielfranken umgerechnet. Haben z.B. 56.62% der Stimmberechtigten ja gestimmt. Dann wird Ihnen pro Ja-Aktie 56.62 Spielfranken ausgezahlt. Wichtig: Der Wert der Aktie am Abstimmungssonntag ist nicht abhängig vom letzten gehandelten Preis, sondern ausschliesslich vom Wahl- bzw. Abstimmungsergebnis. (Daraus bilden wir den Schlusskurs.)

8 Hier können Sie Neuigkeiten aus der Politik und Gesellschaft nachlesen, die mit der Vorlage zu tun haben. Diese sollen Ihnen helfen, sich das Bild über den möglichen Wahl- bzw. Abstimmungsausgang zu verbessern.



1 Hier erhalten nochmals eine Übersicht über Depot- und Marktdaten. Sie sehen Ihr verfügbares Geld, Ihre verfügbaren Aktien, die höchste Nachfrage, das tiefste Angebot und der letzte Kurs.

2 Hier können Sie die Limite und das Volumen eingeben. Ebenso können Sie die Aktie oder den Auftragstyp ändern. Wichtig: Sie können keine Verkaufsaufträge platzieren, wenn Sie bereits einen Kaufauftrag der gleichen Aktie hängig haben, die theoretisch einen Handel auslösen würde. (Und umgekehrt!) Sie dürfen sich selber als keine Aktien verkaufen oder von Ihnen kaufen. Löschen Sie vorher den hängigen Auftrag in der Depotübersicht. Beachten Sie, dass Sie auch Aufträge aufgeben können, die nicht ausführbar sind. Z.B. wenn Sie nicht das nötige Geld oder die nötigen Aktien besitzen. Diese Aufträge werden zwar entgegengenommen aber erst dann ausgeführt, wenn sich etwas an ihrem Kapital geändert hat. Leerkäufe und -verkäufe sind nicht möglich.

3 Hier erscheinen Warnungen oder Fehlermeldungen, sobald etwas mit dem Auftrag nicht stimmt.

10.3.4 Depotseite

Hier sehen sie, welche Möglichkeiten Sie auf der Depotseite haben:

Aktienbestand: 1

	Aktie	Anzahl	Kurs	aktueller Wert	Veränderung
verkaufen 3	Genmoratorium - Ja	300	54	16200	
verkaufen	Genmoratorium - Nein	410	45.88	18810.8	
verkaufen	Ladenöffnungszeiten - Ja	100	54	5400	
verkaufen	Ladenöffnungszeiten - Nein	200	40	8000	
verkaufen	Nationalratswahlen - SVP	190	28	5320	
verkaufen	Nationalratswahlen - SP	100	23	2300	
verkaufen	Nationalratswahlen - FDP	200	14	2800	
verkaufen	Nationalratswahlen - CVP	100	8.5	850	
verkaufen	Nationalratswahlen - Grüne	100	10	1000	
verkaufen	Nationalratswahlen - Andere	80	13	1040	
kaufen 4	Flüssige Mittel			37030.19	2
			Total:	98750.99	-1.25%

1 Hier bekommen Sie eine Übersicht über ihr gesamtes Kapital. Sie sehen hier eine Aufzählung all Ihr erworbener Aktien. Ebenso sehen Sie hier Ihre frei verfügbaren Spielfranken.

2 Hier sehen Sie, wie stark sich Ihr Kapital seit Spielbeginn verändert hat. Beachten Sie, dass hier die momentanen Marktkurse als Bewertungsmassstab genommen wurden. Eine Portfolio hat unabhängig vom Markt immer den Wert von 100 Spielfranken. Sie können ein Portfolio immer über die Bank auf der Handelsübersichtseite verkaufen und erhalten so jederzeit 100 Spielfranken zurück.

3 Über diese Links können sie Verkaufsaufträge von Ihren Aktien aufgeben.

4 Über diesen Link können sie Kaufaufträge über die momentan gehandelten Aktien aufgeben.

Hängige Aufträge: 1

Aktie	Anzahl	Limite	Vorgang	2
Genmoratorium - Ja	555	53.45	Kaufen	löschen
Genmoratorium - Ja	100	53.55	Kaufen	löschen
Genmoratorium - Ja	100	55.5	Verkaufen	löschen
Genmoratorium - Ja	50	56	Verkaufen	löschen
Genmoratorium - Ja	100	56.11	Verkaufen	löschen
Genmoratorium - Ja	100	58	Verkaufen	löschen
Genmoratorium - Ja	100	58.5	Verkaufen	löschen
Genmoratorium - Ja	1000	59	Verkaufen	löschen
Genmoratorium - Nein	1000	41	Kaufen	löschen
Genmoratorium - Nein	500	45	Kaufen	löschen
		Seite	1 →	

1 Hier erhalten eine Übersicht über alle hängigen Aufträge. Beachten Sie, dass evtl. nicht alle Aufträge ausführbar sind. Wenn sie Aufträge platziert haben, die entweder ihr verfügbares Geld oder ihre verfügbaren Aktien übertreffen, dann werden diese erst dann ausgeführt, wenn sie wieder genügend freies Kapital haben oder die Aktien nachträglich erworben haben.

2 Über diesen Link können Sie jederzeit hängige Aufträge löschen.

Kontobewegungen: 1

Datum	Vorgang	Aktie	Anzahl	Preis	Betrag
18.10.2005	Verkaufen	Nationalratswahlen - SVP	10	28	280
18.10.2005	Kaufen	Genmoratorium - Nein	10	45.88	458.8
13.10.2005	Verkaufen	Nationalratswahlen - CVP	100	8.5	850
13.10.2005	Verkaufen	Nationalratswahlen - Grüne	100	10	1000
13.10.2005	Kaufen	Ladenöffnungszeiten - Ja	100	55	5500
13.10.2005	Verkaufen	Genmoratorium - Ja	100	49.99	4999
13.10.2005	Verkaufen	Ladenöffnungszeiten - Nein	50	40	2000
13.10.2005	Verkaufen	Ladenöffnungszeiten - Nein	100	40	4000
13.10.2005	Verkaufen	Nationalratswahlen - SP	100	23	2300
13.10.2005	Verkaufen	Ladenöffnungszeiten - Ja	100	56	5600
			Seite	1 →	

1 Hier sehen Sie eine Übersicht über alle vergangen Kontobewegungen. Beachten Sie, dass Sie in dieser Version nur die Kontobewegungen sehen, die Sie direkt über den Markt abgewickelt haben. Portfoliozukäufe und -verkäufe dagegen, werden (noch) nicht nachgeführt.

10.4. Theoretische Abhandlung der Börse aus der PA2

Wir haben hier zwei Kapitel aus der PA2 über die Funktionsweise eines Auftragsbuch und der Eröffnungsregeln an der Börse.
Obwohl es in unserem System keine Eröffnung gibt, weil die Börse 24 Stunden läuft, haben wir der Vollständigkeit die Eröffnungsregeln beigelegt.

10.4.1 Das Auftragsbuch

Das zentralste Element in der Börse ist das Auftragsbuch. Ohne Auftragsbuch läuft bei einer modernen Börse nichts. Im Auftragbuch stehen alle Ask- und Bid-Aufträge (bzw. Geld- und Brief-Aufträge) drin. Es gibt ein Auftragbuch pro bestimmte Aktie und Handelsplatz. Nehmen wir also an, die Firma *Escargots du mars, SA.* sei börsenkotiert and der Börse SWX und NYSE. Dann führen beide Börsen je ein Auftragsbuch für die Aktien der Firma *Escargots du mars, SA.*

Die Idee an einem Auftragsbuch ist, dass alle Käufe und Verkäufe einer Aktie über denselben Kanal abgewickelt werden und dass immer der Käufer mit dem höchsten Angebot bzw. der Verkäufer mit dem billigsten Preis den Zuschlag kriegt. Aber nur dann, wenn sie sich mit dem Preis auch wirklich treffen. Ein Verkäufer, der für eine Aktie wenigstens CHF 15.- verlangt und ein Käufer der für die Aktie höchstens CHF 14.99 bezahlen will, werden niemals zu einem Abschluss kommen. Darüber hinaus kennen sich die beiden auch nicht, d.h. sie können auch nicht direkt miteinander verhandeln. Die Börse anonymisiert die Aufträge für Aussenstehende. Dagegen wird ein Verkäufer der für seine Aktie wenigstens CHF 10.- verlangt und ein Käufer, der bis zu CHF 20.- bereit ist dafür zu zahlen, alle mal einig. Ob diese zu einem Abschluss kommen ist nämlich fraglich. Was passiert mit einem Käufer, der nämlich bereit ist auch CHF 15.- dafür zu zahlen? Bekommt dieser den Zuschlag, oder doch der andere?

Das Auftragsbuch ist so gestaltet, dass diese Frage ganz automatisch beantwortet wird. Um zu sehen, wie das genau abläuft, betrachten wir nun das Auftragsbuch und ein paar Fälle dazu:

Hier sehen wir, wie ein Auftragsbuch für die Aktie *Escargots du mars, SA.* aussehen könnte am 30.5.2005 um 10:28 Uhr:

Verkäufer			
Preis	**Vol**	**Datum & Zeit**	
11.90	200	30.05.2005	10:25:15
11.80	234	30.05.2005	10:26:12
11.70	5089	30.05.2005	10:27:30
11.60	6457	30.05.2005	10:27:11
11.50	65	30.05.2005	10:26:55
11.40	678	30.05.2005	10:25:12
11.30	100	30.05.2005	10:26:22
11.20	2038	30.05.2005	10:26:52
11.10	222	30.05.2005	10:26:23
11.00	4576	30.05.2005	10:25:16

30.05.2005	10:25:44	2372	**10.90**
30.05.2005	10:26:54	234	**10.80**
30.05.2005	10:26:12	4356	**10.70**
30.05.2005	10:27:33	200	**10.60**
30.05.2005	10:26:23	210	**10.50**
30.05.2005	10:25:43	395	**10.40**
30.05.2005	10:26:05	9988	**10.30**
30.05.2005	10:26:13	674	**10.20**
30.05.2005	10:25:07	3904	**10.10**
30.05.2005	10:25:29	900	**10.00**
Datum & Zeit		**Vol**	**Preis**
Käufer			

Auf der linken Seite sehen wir zusammengefasst alle Käufer und auf der rechten Seite alle Verkäufer. Die Volumen wurden pro Preis zusammengefasst. D.h. pro Preis sind es mehrere Aufträge von verschiedenen Händlern. Die Preise folgen aber in Schritten von Rp. 10.- Das ist aktien-, auftragsbuch- und handelsplatzspezifisch. Die Amerikaner rechnen z.B. in Schritten von 1/8 oder 1/16. So wie dieses Auftragsbuch aufgebaut ist, kommt momentan kein Handel zu Stande.

Bsp 1) Es kommt ein Käufer, der möchte 100 Aktien zu 10.80 kaufen

Sein Auftrag wird einfach hineingehängt und das Volumen wird grösser. Da sein Auftrag aber einen Zeitstempel trägt, haben alle Käufer, die auch für 10.80 kaufen wollen vorher Anspruch, Aktien zu kaufen.

			...	...	...	...
			11.20	2038	30.05.2005	10:26:52
			11.10	222	30.05.2005	10:26:23
			11.00	4576	30.05.2005	10:25:16
30.05.2005	10:25:44	2372	**10.90**			
30.05.2005	10:26:54	334	**10.80**			
30.05.2005	10:26:12	4356	**10.70**			
30.05.2005	10:27:33	200	**10.60**			
...	...	...	...			

Das Volumen wurde beim Preis von 10.80 um 100 grösser.

Bsp. 2) Es kommt ein Händler, der will 100 Aktien für 10.80 verkaufen.

Jetzt werden nicht die 10.80-Aufträge ausgelöst, weil der Verkäufer mit dem Auftrag nicht sagt, dass er sie für exakt 10.80 verkaufen möchte, sondern dass er sie für wenigstens 10.80 verkaufen will. Da gibt es aber ein Volumen von 2›372 von Aufträgen, die auch bereit sind dafür 10.90 zu zahlen. Diese bekommen nun den Zuschlag, weil sie mehr dafür bieten. Es kommt ein Abschluss zu Stande und zwar für 10.90!

			...	...	...	...
			11.20	2038	30.05.2005	10:26:52
			11.10	222	30.05.2005	10:26:23
			11.00	4576	30.05.2005	10:25:16
30.05.2005	10:25:44	2272	**10.90**			
30.05.2005	10:26:54	334	**10.80**			
30.05.2005	10:26:12	4356	**10.70**			
30.05.2005	10:27:33	200	**10.60**			
...	...	...	...			

Bei den 10.90-Aufträgen wurde das Volumen um 100 minimiert.

Abschluss wurde vorgenommen: 100 Aktien für 10.90 am 30.5.2005 um 10:29:15 Uhr auf Ask.
(„Ask“ deshalb, weil der Abschluss ein Verkäufer ausgelöst hat.)

Der Kurs ist der letztgehandelte Preis. Der Aktienkurs steht also nun auf 10.90!

Bsp 3) Es kommt ein Händler, der will 5’000 Aktien kaufen für den Preis von 11.00.

Im Auftragsbuch stehen aber nur 4›576 Aktien zum Verkauf für 11.00. Diese werden auch ausgelöst. Was passiert aber mit den restlichen 424 Aktien? Diese werden einfach zu einem Bid-Auftrag umgewandelt.

			...	...	...	...
			11.30	100	30.05.2005	10:26:22
			11.20	2038	30.05.2005	10:26:52
			11.10	222	30.05.2005	10:26:23
30.05.2005	10:29:55	424	**11.00**			
30.05.2005	10:25:44	2372	**10.90**			
30.05.2005	10:26:54	334	**10.80**			
30.05.2005	10:26:12	4356	**10.70**			
...	...	...	...			

Abschluss wurde vorgenommen: 4'576 Aktien für 11.00 am 30.5.2005 um 10:29:55 Uhr auf Bid.
(„Bid" deshalb, weil der Abschluss ein Käufer ausgelöst hat.)

Bsp. 4) Ein Käufer will 300 Aktien kaufen für den Preis für 11.50

Hier passiert das Ganze analog wie beim Beispiel 2). Der Auftrag sagt nämlich aus, dass der Käufer bereit ist, für die Aktien maximal 11.50 auszugeben. Da es aber viele Aufträge gibt, die billiger sind, kommen die billigsten Aufträge zum Zug.

Er kauft also erstens 222 zu 11.10 und dann noch 78 zu 11.20. Das Auftragsbuch sieht danach so aus:

			...	...	...	...
			11.40	678	30.05.2005	10:25:12
			11.30	100	30.05.2005	10:26:22
			11.20	1960	30.05.2005	10:26:52
30.05.2005	10:29:55	424	**11.00**			
30.05.2005	10:25:44	2372	**10.90**			
30.05.2005	10:26:54	334	**10.80**			
30.05.2005	10:26:12	4356	**10.70**			
...	...	...	...			

Die Verkaufsaufträge von 11.10 sind komplett verschwunden. Es hat momentan zwischen Ask und Bid eine grössere Lücke, was aber vorkommen kann. Sie hält solange, bis wieder ein Händler einen (Kauf- oder Verkaufs-) Auftrag von 11.10 aufgibt, der dann einfach hineingehängt wird, da kein Abschluss bei diesem Preis zu Stande kommen kann. Bei den Aufträgen von 11.20 wurde das Volumen um 78 minimiert.

Abschluss wurde vorgenommen: 222 Aktien für 11.10 am 30.5.2005 um 10:30:17 Uhr auf Bid.

Abschluss wurde vorgenommen: 78 Aktien für 11.20 am 30.5.2005 um 10:30:17 Uhr auf Bid.

Der Aktienkurs steht nun auf 11.20.

Ich hoffe, man kann die weitere Logik aus den Beispielen herauslesen.

10.4.2 Eröffnung

Das Auftragsbuch regelt den Handel problemlos, solange die Aufträge sequentiell hineinkommen. Bei der echten Börse (und deshalb auch in unserer Implementierung berücksichtigt) kommen Aufträge auch vor und nach Börsenschluss hinein. Das Problem ist nun, da sich mehrere Kauf- und Verkaufsaufträge überschneiden können. Und wenn die Börse eröffnet, dann muss dieses Problem vom Auftragsbuch zuerst mal gelöst werden.

So sehen z.B. die Aufträge aus:

				...	...	...	...
				11.40	65	30.05.2005	23:34:12
				11.30	80	30.05.2005	23:34:00
30.05.2005	23:34:31	200	**11.20**	**11.20**	50	30.05.2005	23:33:00
30.05.2005	23:36:22	150	**11.10**	**11.10**	100	30.05.2005	23:39:09
30.05.2005	23:34:35	200	**11.00**	**11.00**	200	30.05.2005	23:32:50
30.05.2005	23:31:02	50	**10.90**				
30.05.2005	23:34:09	70	**10.80**				
...	...	...	...				

Jetzt könnten die 11.20-Kaufaufträge mit den 11.20-Verkaufsaufträgen abgeschlossen werden und die 11.10-Kaufaufträge mit den 11.10-Verkaufsaufträgen, etc. Oder man könnte die 11.20-Kaufaufträge mit den 11.00-Verkaufsaufträgen abschliessen, etc.

Die Frage ist nur, wie soll das das Auftragsbuch lösen, so dass die meisten Abschlüsse zu Stande kommen und sich keiner am Schluss noch benachteiligt vorkommt.

Dazu geht das Auftragsbuch so vor: Es ermittelt einen Eröffnungskurs, der dann für alle gilt. Dieser wird so berechnet:

Zuerst einmal bildet man die Summe aller Volumen der Kaufaufträge, die sich mit den Verkäufern schneiden und multipliziert diese mit dem Preis:

Also z.B.:

Total = 11.00 • 200 + 11.10 • 150 + 11.20 • 200 = 6105

Das Ganze auch bei den Verkäufern:

Total = 11.20 • 50 + 11.10 • 100 + 11.00 • 200 = 3879

Dann zählt man beide zusammen und erhält: 9975

Der Kurs ist nun diese Summe geteilt durch alle Volumen der Aufträge die sich überschneiden:
Kurs = 9975 : (200+150+200+50+100+200) = 11.0833

Den Kurs runden wir nun mathematisch in Rp. 10.- Einheiten:
Eröffnungskurs = 11.10

Jetzt können wir nachsehen, welche Aufträge also mit diesem Eröffnungskurs abgeschlossen werden können:

				...	...	...	...
				11.40	65	30.05.2005	23:34:12
				11.30	80	30.05.2005	23:34:00
30.05.2005	23:34:31	200	**11.20**	**11.20**	50	30.05.2005	23:33:00
30.05.2005	23:36:22	150	**11.10**	**11.10**	100	30.05.2005	23:39:09
30.05.2005	23:34:35	200	**11.00**	**11.00**	200	30.05.2005	23:32:50
30.05.2005	23:31:02	50	**10.90**				
30.05.2005	23:34:09	70	**10.80**				
...	...	...	...				

Alle grünen Aufträge werden nun miteinander ausgelöst und zwar zu einem Kurs von 11.10!
Nur die 50 Aktien des Kauf-Auftrages zu 11.10 mit dem jüngsten Zeitstempel bleiben stehen.

So sieht das Buch gleich nach der Eröffnung aus:

				...	...	...	...
				11.40	65	30.05.2005	23:34:12
				11.30	80	30.05.2005	23:34:00
				11.20	50	30.05.2005	23:33:00
30.05.2005	23:36:22	50	**11.10**				
30.05.2005	23:34:35	200	**11.00**				
30.05.2005	23:31:02	50	**10.90**				
30.05.2005	23:34:09	70	**10.80**				
...	...	...	...				

10.5 Sitzungsprotokolle

09.09.05 - 14.00 Uhr – Sitzungszimmer Ierax

Protokoll erstellt von: GS

Anwesend:
Ferri Reto
Rieser Micha
Scrugli Giancarlo

Ablauf:

(1) Besprechung der Resultate der 1. Iteration (Elaboration):

Disziplin:	Artefakte:	Bemerkungen:
Analyse	-Use Case „Handel“ -Use Case Diagramm	-dieser UseCase wurde in der 1. Iteration realisiert; Er wurde fully dressed dokumentiert (alle Alternativen dieses Usecases sind beschrieben) -ok
Design	-Domain Model -Kollaborations-diagramm -Klassendiagramm	-ist von Hand gezeichnet. Wurde nach „Larman“ in kurzer Zeit realisiert -beschränkt sich auf die Hauptoperation im Use case Handel Herr Ferri hat uns geraten für die Dokumentation ein vollständigeres Kollaborationsdiagramm zu präsentieren. -(ok)
Implementierung	-Präsentation Resultat 1. Iteration	-Präsentation Ok. Es besteht allerdings noch ein Problem mit der Darstellung der Aufträge mit gleichem „Limit“(Preis). Zudem besteht der Typ „DateTime“ im Grid nur aus dem Datum ohne Zeit. Das führt zum Problem, dass wenn zwei Aufträge im Orderbook bestehen und ein Auftrag ausgeführt wir, der die Aufträge nicht vollständig auslöst, es nicht klar ist, welcher Auftrag und warum zuerst ausgelöst wird. Später sollen die Aufträge zusammengefasst werden und es erscheinen die Aufträge mit gleichem Preis im View als ein Auftrag. Die Business-Logik im Orderbook wurde aus der Projektarbeit 2 übernommen und funktioniert bereits.
Test	-Test der Funktionalität des Börsenbuchs über den View - ok	Es folgen noch White- und Blackbox-Tests

(2) Thema Risiken minimieren

Im Gespräch wurden die zwei Hauptrisiken festgehalten.
Das erste Risiko besteht in der Anbindung der Businesslogik an eine Datenbank. Das zweite Risiko ist der Einsatz, der uns noch wenig bekannten Technologie ASP.NET.

Zur Datenbank:

Wir haben versucht die Daten vom Handel persistent in einer Access Datenbank abzuspeichern. Leider sind wir an den einfachsten Funktionen, wie Auftrag löschen/hinzufügen bereits gescheitert. Wir haben dann gestern Donnerstag entschieden, die Frage der Datenbank auf die nächste Iteration zu verschieben. Wir haben zudem Herrn Ferri gefragt, ob er uns eine geeignete Lösung vorschlagen kann.
Das bei der Ierax auch auf .NET Software entwickelt wird, haben wir konkret gefragt ob wir die Lösung im Unternehmen uns anschauen können.
Anwort von Herrn Ferri: Was sie für den DCI (DMS) im Einsatz haben, ist nicht geeignet für unsere Lösung. Herr Ferri hat uns dann einige Vorschläge gemacht.
Vorschläge:
MS-SQL (Download unter CodeZone).
Hier sollte die Unterstützung von .NET gewährleistet sein.
MSDE (Microsoft Developer Environment).
Ist im Gegensatz zu MS-SQL gratis.
Micha Rieser: Wir sind so Microsoft lastig in dieser Arbeit
Reto Ferri: Das ist ein Entscheid, den wir früh schon gefällt haben und natürlich seine Konsequenzen mit sich trägt.

Zu ASP.NET:

Um das Risiko möglichst rasch zu minimieren, wäre es von Vorteil ASP.NET möglichst früh im Projekt in Angriff zu nehmen. Beim durchgehen des Zeitplans ist und aufgefallen, dass ASP.NET erst am Schluss geplant ist. Der Zeitplan wird zu Gunsten von ASP.NET wahrscheinlich noch angepasst.

Beschlüsse:
1. Wir bleiben auf der Microsoft “Schiene“.

Bemerkungen von Herrn Ferri:
In der Dokumentation soll ein Kapitel, der das Projektmanagement beschreibt, unbedingt vorhanden sein.
Ein “Reservationsmechanismus“ für die Aufträge muss vorhanden sein. Das heisst: Was wänn zwei User unabhängig voneinander den identischen Auftrag, gleichzeitig aufgeben?

Ende der Sitzung: 14.50 Uhr

Nächste Sitzung: 16.09.2005 (13:30 bei der Ierax)

16.09.05 - 13.30 Uhr – Sitzungszimmer Ierax

Protokoll erstellt von: GS

Anwesend:
Ferri Reto
Rieser Micha
Scrugli Giancarlo

Ablauf:
(1) Abschlusspräsentation vor den Experten

Herr Ferri informierte uns über unsere Abschlusspräsentation. Die Präsentation findet am Montag 14.11.2005 im e425 um 13:00 statt. Für die Präsentation stehen uns rund 60min zur Verfügung. Üblicherweise, wird in diesen 60min ca. 30min. – 45min. Präsentiert (inklusive kurze Demonstration) der Rest der Zeit ist für Fragen und Diskussionen offen.

An der Abschlusspräsentation sind Anwesend der Dozent, und zwei Experten. Beide Experten kennen die Arbeit, der eine Detailliert, der andere hat die Dokumentation nur überflogen. Fragen können aus dem Kontext gestellt werden, d.h. auch während der Präsentation. Üblicherweise wird auf schwächen oder Unklarheiten in der Dokumentation hingewiesen. (Mit anderen Worten: Es muss alles sauber und klar in der Dokumentation beschrieben werden)

(2) Thema Risiken minimieren

Die festgestellten Risiken aus der letzten Itaration, sprich: Datenbank und ASP.NET, konnten nicht minimiert werden. Es wurde ein SQL Server installiert, leider aber noch ohne Erfolg. Am Samstag wird ein weiterer Anlauf vorgenommen, dieses Mal versuchen wir es mit MySQL. Wir nehmen ein weiteres DBMS in Betracht, weil wir damit das Risiko vermeiden wollen, mit MS SQL zu viel Zeit zu verlieren.
ASP.NET wird in einer späteren Iteration betrachtet.

(3) Kurze Demo der Resultate in der 2. Iteration

Wir haben Herrn Ferri erklärt inwieweit sich unsere Prognosebörse im Kern von einer echten Börse (vergl. SWX) neu unterscheidet. Wir haben in dieser Woche festgestellt, dass ein Intelligentes System für eine Prognosebörse eher zutrifft. Zudem war es derselbe Ansatz für vergangene Prognosebörsen (vergl. Universität von IOWA 1988 – Präsidentschaftswahlen).
Das Kollaborationsdiagramm wirkt auf Herrn Ferri als „zu kompliziert".

(4) Die nächste Iteration

Nach dem Test der sehr komplexen Businesslogik, wird die Statistik in Angriff genommen. Am ende dieser Iteration soll es möglich sein, Kursverläufe zeichnen zu lassen.

Beschlüsse:
Es wurden keine weiteren Beschlüsse getroffen.

Ende der Sitzung: ca. 14.10 Uhr

Nächste Sitzung: 23.09.2005 (14:00 bei der Ierax)

23.09.05 – 14.00 Uhr – Sitzungszimmer Ierax

Protokoll erstellt von: GS

Anwesend:
Ferri Reto
Rieser Micha
Scrugli Giancarlo

Ablauf:
(1) Neudefinition der 3. Iteration: Code-Review + Redesing

Durch die Einführung einer MSSQL Datenbank, hat sich bei uns ein Code-Review und Redesign der Businesslogik aufgedrängt. Mit der Einführung der Datenbank haben sich nämlich Anforderungen an das System z.T. verändert.
Nach der Einbindung der Datenbank an unsere Businesslogik, haben wir den Kern mit Depots erweitert.
Aufgrund des Redesign unseres Kerns, haben wir das Testen (White- bzw. Blackbox Test) nicht durchgeführt. Haben wir mal einen stabilen Kern, werden wir es auch austesten.
Das Neue Design sieht folgende Änderungen vor:
- Bank macht keine Transaktionssteuerung (neu in eigener Klasse)
- Bank und Market greifen auf Depots bzw. Orderbooks über den MarketObjectManager
- Der MarketObjectManager verwaltet den Speicher und stellt dem Market bzw. Bank die Referenzen zum Depot bzw. Orderbook zur Verfügung.

(2) Neu ist eine MSSQL DB im Einsatz + Transaktionen wurde eingeführt

Die Datenbank ermöglicht es uns, die Daten persistent abzuspeichern. Damit die Daten auf der Datenbank aber konsistent bleiben, ist die Einführung von Transaktionen unabdingbar. Diese Transaktionssteuerung haben wir momentan der Klasse Bank übergeben. Später wird das aber eine eigene Klasse (z.B. TransactionController) übernehmen.

(3) Lasttest + MarketObjectManager

Nach Anbindung der Datenbank an unsere Businesslogik, haben wir einen Lasttest durchgeführt. Wir haben dabei, die Datenbank mit 10›000 zufälligen Usern und Orderbooks gefüllt. Beim starten unseres vorläufigen Views wurde aus den Daten aus der Datenbank 10›000 Depots bzw. 10›000 Orderbooks instanziert. Uns ist dann bewusst geworden, dass unser System, einerseits viel Ressourcen braucht (200MB Arbeitsspeicher) und andererseits alle auch nicht aktiven Depots und Orderbooks während der gesamten Laufzeit im Speicher bleiben.
Der MarketObjectManager soll genau dieses Problem lösen und die nicht aktiven Objekte für den GarbageCollector frei geben. Der MarketObjectManger kann also bei vorhandenem Objekt einfach die Referenz weitergeben oder das Objekt falls noch nicht vorhanden instanzieren.
Der Mechanismus im MarketObjectManager ist vergleichbar mit der LRU-Zeiger, der beim durchgehen auf die Objekte einen Zähler runterzählt, wenn das Objekt nicht referenziert wird.
Herr Ferri hat uns dabei aufmerksam gemacht, dass es zwei bessere Ansätze gibt, als das LRU Prinzip.
nicht alle Zähler runterzählen, sondern den Zähler in Objekten hochzählen, die referenziert werden.
(Die noch bessere Ansatz von Reto Ferri „himself“) Referenzen in ein Array ablegen und wie ein Stack behandeln. D.h. wird ein Objekt referenziert, wird es „unten“ entfernt und oben an erster Stelle auf den „Stack“ gelegt.
Herr Ferri hat uns weiter erzählt, dass dieses „Speicher-Problem“ typischerweise bei Transaktionsorientierten Systemen vorkommt. In Java gibt’s eine Lösung in J2EE und .NET bietet auch eine Lösung zum Problem unter den Enterprise Component. (COM+ Objectpools)

Allerdings haben wir im Gespräch festgestellt, dass wir die Speicherverwaltung einfacher mit WeakReferences und einer DirtyList (mit Referenzen zu allen geänderten Depots) realisieren können.

(4) Timestamp – oder DateTime?

Der Einsatz von DateTime anstatt der eigens implementierten TimeStamp wird nochmals in betracht gezogen, da der TimeStamp nicht mehr erfüllt, als das DateTime nicht schon kann.

(5) Logfiles werden erstellt durch die neue Protokoll klasse

Micha hat eine Klasse Protokoll geschrieben, die uns Ausgaben vom Programm auf File (Textfile) schreibt. Herr Ferri hat uns darauf hingewiesen, dass diese Funktionalität .NET schon anbietet. Es befindet sich unter System.Diagnostics.

(6) Dynamische Fehler mit DataViews – Bug in .NET?

Micha hat beim testen festgestellt, dass die DataViews, welche wir beim löschen von Orders einsetzten, z.T. nicht richtig gefüllt wurden. Herrn Ferri, ist dieses Problem nicht bekannt. Er hat uns aber darauf hingewiesen, ob der Einsatz von DataSet jetzt mit der Datenbank überhaupt noch nötig ist. Wir haben noch nicht entscheiden, ob wir auf DataSet verzichten möchten, auf DataView aber bestimmt.

(7) ASP.NET wurde auch in Angriff genommen

ASP.NET also noch bestehendes Risiko wurde von Giancarlo in dieser Woche in Angriff genommen. Wir sind im ASP.NET auf dem Stand, User über eine Datenbank-Abfrage zu authentisieren und mit den Daten aus der Session, die Daten der User auf der Webseite anzuzeigen. Giancarlo hat zudem den Umgang mit Grids auf der Webseite untersucht und getestet. Sicherheitsaspekte wie, sicheres Einloggen über ASP.NET wurde auch abgeklärt, ist aber noch nicht getestet.
Falls nicht weitere Anforderungen hinzukommen, können wir für unsere DA mit diesem Grundgerüst schon sehr weit kommen.

Beschlüsse:
Wir haben beschlossen ein Redesign unseres Kerns zu machen und die Anstösse und Ideen aus der heutigen Sitzung einfliessen zu lassen. Ob alles wirklich realisiert wird (Bspw. Einsetzten der DateTime anstatt TimeStamp oder das Loggen mit Hilfe der Trace Klasse aus System. Diagnostics) ist nicht zwingend. Herr Ferri hat diese Entscheidungen uns überlassen. Was wir aber sicher implementieren möchten ist die Idee der WeakReferences und DirtyList für eine effiziente Speicherverwaltung.

Ende der Sitzung: ca. 15.20 Uhr

Nächste Sitzung: 30.09.2005 (14:00 bei der Ierax)

30.09.05 – 14.30 Uhr – Sitzungszimmer Ierax

Protokoll erstellt von: GS

Anwesend:
Reto Ferri
Micha Rieser
Giancarlo Scrugli

Ablauf:
(1) Re-Coding der gesamten Arbeit (Das Gesamte wird generisch)

In dieser Iteration haben wir die gesamte Businesslogik „Re-Coded". Da wir explizit das gesamte neu geschrieben haben, sprechen wir hier von „Re-Coding" und nicht von „Re-Factoring".
Wir haben dabei auch festgestellt, dass für die Rekursion lediglich die ASK-Aufträge weiter verarbeitet werden können. Darauf hin haben wir die Auftragsbearbeitung komplett neu konzipiert und geschrieben. Das wir nur ASK-Aufträge berücksichtigen müssen für die Rekursion, ist auch daraus entstanden, weil wir Aufträge, vom gleichen User, die sich selber auslösen nicht erlauben. Das hat den Ablauf in der Businesslogik klarer gemacht.
Für die Präsentation hat uns Herr Ferri ans Herz gelegt, die korrekt ablaufende Rekursion wenn möglich an der Präsentation zu zeigen.

(2) Weak References

Aus der letzten Sitzung aus, haben wir den Gebrauch von Weak References für die Verwaltung von Objekten im Speicher untersucht. Das Problem war ursprünglich, dass bei sehr vielen Depots (10›000) und Ordebooks (10›000) die Applikation sehr Speicherintensiv ist.
Für unseren Gebrauch haben sich aber Weak References nicht als ideale Lösung herausgestellt, aufgrund des Mehraufwandes bei der Verwaltung der Keys der Objekte im Speicher, die per Weak Reference referenziert werden.
Für die Objekte-Verwaltung im Speicher haben wir nun eine eigene „MangedListe" implementiert, die nichts anderes ist als eine „ArrayListe", die beliebig anwachsen kann und jeweils nach einer Transaktion wieder „zurechgeschnitten" wird. Somit vermeiden wird das unnötig viele Objekte im Speicher gehalten werden.

(3) ObjectManager

Für die Instanzierung der einzelnen Objekte haben wir uns „Reflection" zur Hilfe genommen. Herr Ferri meinte, dass es schön sei zu erfahren, dass es mit „Reflection" geht, aber es besser sei mit dem „Factory Pattern" zu arbeiten. Zudem meinte er, dass wir mit „Reflection" daran gebunden sind, die Klasse nach einer gewissen „Signatur" zu instanzieren. Die „Sgnatur", kann beispielsweise der „Namespace" und der Klassenname sein. Das „Factory-Pattern" gibt da mehr Freiheit.
Grundsätzlich meinte Herr Ferri, wir sollten „Statische Methoden" und das „Singleton-Pattern" vermeiden.
Momentan verwaltet der „ObjectManager" die Objekte im Speicher mit Hilfe der „Manged List". Die Instanzierung der Objekte muss allerdings nochmals überdacht werden. Wir werden den Einsatz vom „Factory-Pattern" auch in Betracht ziehen.

(4) GUID

GUIDs werden im Windowsumfeld im Zusammenhang mit Active Directory eingesetzt. Für unseren Gebrauch sind GUIDs zu gross und mächtig. Wir bleiben bei unseren eigenen Unique ID.

(5) Klasse „TimeStamp“ wurde entfernt

Die Klasse „TimeStamp“ haben wir entfernt und brauchen nun als Zeitstempel die im C# schon vorhandene „DateTime“ Klasse. (DateTime.Now.Ticks)

(6) „Structs“ eliminiert

Ursprünglich waren unsere Orderbooks, Depots etc als „Struct“ definiert. Wir haben allerdings Probleme gehabt beim entwickeln. Herr Ferri hat uns diesbezüglich darauf hingewiesen, dass „Structs“, wie Wertetypen behandelt werden. Das bedeutet, dass auch bei Structs „boxing“ und „unboxing“ existiert, dass wiederum heisst, dass „Struct“ sowohl auf dem Heap als auch auf dem „Stack“ zu liegen kommen. Dieses Problem, ist der Grund unserer Probleme mit der Arbeit mit „Structs“.

(7) „DataSet“ entfernt

Ein Beschluss aus der letzten Woche war, nicht mehr mit „DataSet“ zu arbeiten. Das „DataSet“ bietet für unsere Arbeit keine echten Vorteile, also haben wir auf den „DataReader“ umgestellt. Somit schreiben wir die Daten am Ende einer Transaction direkt auf dem Daten Bank Server. Implizit haben wir das mit dem „DataSet“ auch gemacht. (von verbindungslos® zu verbindungsorentiert)

(8) Klasse „MarketInfo“ instanziert alle Objekte

Die Klasse „MarketInfo“ instanziert alle Objekte und stellt die Referenzen zur Verfügung.
Herr Ferri hat uns darauf hingewiesen, dass nach seiner Meinung nach, in unserem Projekt die typische „transaktionelle Natur“ fehlt.
Ausserdem hat er bemängelt, dass der „DirtyStack“ öffentlich sichtbar ist. Wir sollen den „DirtyStack“ verstecken.

Beschlüsse:
- Auf Hinweis von Herrn Ferri, werden wir den „Singleton“ rausnehmen.
- Wir führen Das „Interface“ „Bean“ ein, die von den nötigen Interfaces erbt um ein Tupel zu speichern, lesen.. etc
- Die Transaktionellen und Statischen Teile des Systems sollen besser getrennt werden.
- Der „TransactionController“ bleibt ein „Singleton“

Bemerkungen von Herrn Ferri:
- Properties werden laut .NET Konvention klein geschrieben.
- Singelton Pattern und statische Methoden vermeiden.
- Referenzen über Interfaces ist besser aus Erfahrung (Low Coupling Prinzip)

Ende der Sitzung: ca. 15.40 Uhr

Nächste Sitzung: 07.10.2005 (14:00 bei der Ierax)

07.10.05 – 14.00 Uhr – Sitzungszimmer Ierax

Protokoll erstellt von: GS

Anwesend:
Reto Ferri
Micha Rieser
Giancarlo Scrugli

Ablauf:
(1) Fixer Kern

Der Kern wurde auf Hinweise von letzter Woche angepasst und ist soweit abgeschlossen.

(2) „Repeater“ und weitere Probleme mit ASP.NET

Bei unseren ersten Gehversuchen mit ASP.NET sind wir auf das „Repeater“-Konstrukt gestossen, das angeblich eine gute Alternative zu „DataGrid“ bietet, da beim „Repeater“ die Darstellung nicht wie beim „DataGrid“ vorgegeben ist. Wir hatten aus unserem „Dummy-View“ aus der ersten Iteration schon klare Vorstellungen, wie Beispielsweise unsere Handelsübersichtseite zu aussehen hat. Wir haben nun also versucht, das ganze in ASP.NET mit Hilfe des „Repeaters“ umzusetzen. Leider mussten wir feststellen, dass so wie wir es gerne hätten, nicht mit dem „Repeater“ machbar ist.
Im neuen Ansatz setzten wir den HTML Code direkt im „Code Behind“ unserer ASP.NET Seite zusammen und senden es als „Response“ auf die Seite in Form eines „Labels“ oder „Literal“ ganz einfach zurück. Hiermit können wir unabhängig von ASP.NET unsere Seiten treu unseren Vorgaben darstellen.
Herr Ferri hat uns von diesem Ansatz gewarnt, weil wir damit versuchen ASP.NET zu umgehen. Weiter warnte er uns, dass dieser Ansatz bestimmt nicht gut bei den Experten ankommen würde. Wir sollten diese Lösung unbedingt nochmals überdenken und versuchen mit der Technologie besser zurecht zukommen. Eine Möglichkeit bieten „Webcontrols“. Falls wir dabei aber bleiben wollen, muss es uns klar sein, dass die Experten solche Entscheide in Frage stellen werden.

(3) Klassendiagramm

Wurde angepasst. Einzig zu diskutieren gab die Farbgebung der Klassen.
Grün: Temporär Transaktionsunabhängige Objekte
Blau: Statische und Transaktionsunabhängige Objekte
Gelb: Temporär Transaktionsabhängige Objekte
Herr Ferri und Micha Rieser haben um die Namensgebung diskutiert, weil für Herrn Ferri die Grün bemalten Objekte auch Transaktionsabhängig sind. Tatsächlich sind sowohl die Grünen also auch die Gelben Objekte nur während einer Transaktion vorhanden: also Transaktionsabhängig. Was Micha aber mit dieser Farbgebung aussagen wollte, ist, dass während ein Gelbes Objekt in jeder Transaktion anders aussieht, die Grünen Objekte immer gleich bleiben. Da für Herr Ferri die Beschreibung irreführend ist und diese Unterscheidung im Grunde genommen nicht nötig ist, hat er uns vorgeschlagen, entweder die Namensgebung anzupassen oder die Unterscheidung zwischen Grün und Gelb nicht zu machen.

Beschlüsse:
-Wir werden unseren Ansatz überdenken und auf den Einsatz von „Webcontrols“ zielen.

Ende der Sitzung: ca. 15.00 Uhr

Nächste Sitzung: 14.10.2005 (14:00 bei der Ierax)

14.10.05 – 14.00 Uhr – Sitzungszimmer Ierax

Protokoll erstellt von: GS

Anwesend:
Reto Ferri
Micha Rieser
Giancarlo Scrugli

Ablauf:
(1) Präsentation des Schlussresultats

Wir haben in der letzten Woche die Seite soweit abgeschlossen mit Berücksichtigung der Erkenntnisse aus der letzten Sitzung. Wir haben jetzt also zum Darstellen der Handelsübersichtseite und Depotseite „Webcontrols" erstellt, die sich im „CodeBehind" und mit Hilfe von „HTMLGenerator" Klassen erstellt und auf der Seite als normaler HTML Code dargestellt werden. Somit nutzen wir die Vorzüge aus ASP.NET („WebControls") und können zusätzlich unsere Darstellung beibehalten.
Zusätzlich haben wir neu die Möglichkeit, die Datenbank lokal auf dem Laptop laufen zu lassen und konnten somit alle Funktionen in der Sitzung zeigen. (z.B. Einloggen, Handeln, Depotansicht etc). Für die Darstellung auf der Webseite, haben wir noch so genannte „ViewBeans" geschrieben, die die nötigen Daten mit Hilfe der schon vorhandenen Businesslogik aus der Datenbank direkt beziehen.
Was noch nicht implementiert wurde sind, die Statistik und ein Administrationstool. Das Administrationstool ist allerdings nicht mehr Teil unserer Aufgabe. Folglich haben wir wenig Zeit dafür geplant. Herr Ferri unterstützt uns in diesem Entscheid und sagte, dass ein Administrationstool bestimmt eine nützliche Sache ist, wir aber dafür nicht die Dokumentation vernachlässigen dürfen und klar Prioritäten setzen müssen.
Wir werden in der nächsten Woche sehen, wie weit wir dabei kommen.

(2) Usability vom „Politmarket"

Da wir darauf zielen, diese Arbeit zu veröffentlichen und es bis zu den nächsten Abstimmungen laufen zu lassen, möchten wir an der Usability noch feilen und das ganze verträglicher für die breite Masse machen.

(3) Betrieben der Webseite auf ZHW Servern?

Um unsere Arbeit zu veröffentlichen benötigen wir eine gewisse Infrastruktur. Wir wollten gerne wissen, ob wir bis am 27.11. die Möglichkeit haben unsere Seite auf einem ZHW Server Live aufschalten können. Herr Ferri hat uns versichert, dass es diese Möglichkeit gibt, er aber nicht genau informiert ist darüber, ob wir so weit über dem Abgabetermin diese Seite laufen lassen können. Herr Ferri hat uns gebeten mit Herrn Hutter dafür direkt Kontakt aufzunehmen.

(4) Weitere betreiben der Webseite auch nach der Diplomarbeit?

Wir wünschen uns diese Seite auch nach der Diplomarbeit zu betreiben, natürlich ohne damit Geld zu verdienen. Herr Ferri hat uns gesagt, dass es sicherlich, die Möglichkeit gibt, wir aber das am Ende nochmals besprechen müssen und allen Falls einen Vertrag abschliessen müssen, da das Resultat dieser Diplomarbeit der ZHW gehört.

Beschlüsse:
Es wurden keine weiteren Beschlüsse getroffen.

Ende der Sitzung: ca. 15.00 Uhr

Nächste Sitzung: 21.10.2005 (14:00 bei der Ierax)

21.10.05 – 14.00 Uhr – Sitzungszimmer Ierax

Protokoll erstellt von: GS

Anwesend:
Reto Ferri
Micha Rieser
Giancarlo Scrugli

Ablauf:
(1) Weiterführen von Arbeiten am Politmarket auch nach dem Abgabetermin

Wir haben bereits den Politmarket auf vwww.politmarket.net „live" geschaltet und möchten für die bereits angemeldeten Benutzer an der Seite weiterarbeiten. Da es sich aber um eine Diplomarbeit handelt, muss noch klar geklärt werden, in wieweit man da weiter arbeiten darf. Herr Ferri hat uns darauf geantwortet, dass wir grundsätzlich nicht weiterarbeiten dürfen an der Diplomarbeit. Wir sollten also darauf achten, dass die Version, die wir dokumentieren auch an der Präsentation vorgestellt wird. Was wir allerdings machen dürfen, ist parallel weiterarbeiten und allenfalls an der Präsentation dann die aktuelle Version auch noch zeigen. Aber Hauptbestandteil der Präsentation soll die dokumentierte Version sein.

(2) Konkrete Erwartungen an Dokumentation

Wir haben Herrn Ferri gefragt, was er n der Dokumentation als ein „must have" sieht.
Folgende Punkte hat er erwähnt:
-Anforderungen
-Analyse
-Desing
-Implementierung reicht auf der CD am besten gleich mit „XML-Doc" (ähnlich Java-Doc)
-Glossar
-Benutzeranleitung
-Anwenderdokumentation
Wir sind davon ausgegangen, dass wir den Code, beschrieben in die Dokumentation nehmen müssen. Er hat uns versichert, dass das nicht nötig sei den Code zusätzlich in der Dokumentation zu beschreiben.

(3) Abgabetermin

Für die Abgabe der Diplomarbeit, haben wir uns auf folgendes geeinigt: Mittagspause bei der Iearax.

(4) Kapitel „Fehler und Verbesserungsvorschläge" in der Dokumentation?

Wir haben nach Abschliessen der Entwicklungsarbeiten und „live" Schaltung unserer Webseite, gewisse Mängel festgestellt und auch schon Lösungsansätze Diskutiert. Anstatt diese jetzt noch auf die schnelle zu korrigieren, wollten wir diese Mängel in einem separaten Kapitel behandeln. Grundsätzlich sei das eine Gute Idee.
Wir werden aber noch sehen, ob und wie viel so ein Kapitel wirklich von sich gibt.

Beschlüsse:
Es wurden keine weiteren Beschlüsse getroffen.

Ende der Sitzung: ca. 15.00 Uhr